VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Nasazení technologie WebRTC s využitím SIP Proxy Kamailio

Deploying WebRTC technology using SIP Proxy Kamailio

2019 Dalibor Zeman

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Dalibor Zeman**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2601R013 Telekomunikační technika

Téma: Nasazení technologie WebRTC s využitím SIP Proxy Kamailio
Deploying WebRTC technology using SIP Proxy Kamailio

Jazyk vypracování: čeština

Zásady pro vypracování:

Technologie WebRTC se v současnosti stala trendem v IP telefonním světě. Komunikace pomocí WebSocketů je již otestovaná a plně funkční v komerčních aplikacích, avšak open-source produkty ještě vyžadují často optimalizaci konfigurace a přídavné moduly pro správnou výměnu komunikace. Cílem bakalářské práce je navrhnout, nakonfigurovat, otestovat a zdokumentovat možnosti komunikace s využitím WebRTC technologie na SIP Proxy Kamailio.

Body zadání:
1. Nastudujte WebRTC, HTML5, komunikace s využitím WebSocketů.
2. Detailně popište způsoby přenosů IP telefonního provozu skrze WebSockety.
3. Prakticky navrhněte testovací topologie a konfigurace WebRTC klientů se SIP Proxy Kamailio.
4. otestujte funkčnost a analýzu zachyceného provozu komunikace.
5. Podrobně zadokumentujte funkční topologie.

Seznam doporučené odborné literatury:

[1] Getting Started with WebRTC, Rob Manson , 2013, ISBN-10: 1782166300.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Filip Řezáč, Ph.D.**

Datum zadání: 01.09.2018
Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2019 .....

Chtěl bych poděkovat zejména svému vedoucímu bakalářské práce, Ing. Filipu Řezáčovi, Ph.D., za poskytnutí nakonec zajímavého tématu a v neposlední řadě za jeho pozitivní přístup a vstřícné konzultace. Dále oceňuji neortodoxní výuku Ing. Jana Rozhona, Ph.D., jehož cvičení mi v prvním semestru tohoto akademického roku poskytla skvělý úvod do světa VoIP.

Abstrakt

Cílem této bakalářské práce je navrhnout testovací topologii pro WebRTC komunikaci s využitím SIP proxy serveru Kamailio a zejména dodat postup, jak byla tato topologie zprovozněna. Zprovoznění zahrnuje konfiguraci Kamailia a WebRTC klientů. Vzhledem k povaze protokolů používaných ve WebRTC je nedílnou součástí návrh vhodného řešení pro zachycení a analýzu SIP komunikace. WebRTC je poměrně nová technologie, která teprve získává pozornost, a ucelené návody k jejímu zprovoznění jsou v této kombinaci steží dostupné. Konfigurace SIP serveru Asterisk pro WebRTC komunikaci je již součástí výuky IP telefonie a tato práce by měla být základem pro obdobné cvičení.

Klíčová slova: WebRTC, SIP proxy, Kamailio, VoIP

Abstract

The objective of this bachelor thesis is to design a testing topology for WebRTC communications using SIP proxy server Kamailio and especially to deliver the instructions, how is this topology made operational. This includes configuration of Kamailio and WebRTC clients. Given the nature of the protocols used in WebRTC, proper solution for capturing and analysis is required. WebRTC is relatively a new technology that still acquires attention. The instructions for making these entities to work are hardly to be found. A configuration of SIP server Asterisk for WebRTC communicaion is already a part of IP telephony classes and this bachelor thesis should serve as a foundation for further analogical practice.

Key Words: WebRTC, SIP proxy, Kamailio, VoIP

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
B2BUA	–	Back-to-Back User Agent
CSR	–	Certificate Signing Request
DNS	–	Domain Name System
DTLS	–	Datagram Transport Layer Security
FQDN	–	Fully Qualified Domain Name
HEP	–	Homer Encapsulation Protocol
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
ICE	–	Interactive Connectivity Establishment
IETF	–	Internet Engeneering Task Force
IP	–	Internet Protocol
JSEP	–	JavaScript Session Establishment Protocol
NAT	–	Network Address Translation
P2P	–	Peer-to-peer
RFC	–	Request For Comment
RTCP	–	RTP Control Protocol
RTCP	–	Real-Time Control Protocol
RTCWeb	–	Real Time Communications on the Web
RTC	–	Real-Time Communications
RTP	–	Real-Time Transport Protocol
SCTP	–	Stream Control Transmission Protocol
SDP	–	Session Description Protocol
SIP	–	Session Initation Protocol
SRTCP	–	Secure Real-Time Control Protocol
SRTP	–	Secure Real-Time Transport Protocol
SSE	–	Server-Send Events
SSL	–	Secure Sockets Layer
STUN	–	Session Traversal Utilities for NAT
TCP	–	Transmission Control Protocol
TLS	–	Transport Layer Security
TURN	–	Traversal Using Relays around NAT
UAC	–	User Agent Client
UAS	–	User Agent Server
UA	–	User Agent
UDP	–	User Datagram Protocol

UI	–	User Interface
URI	–	Uniform Address Identifier
URL	–	Uniform Resource Locator
VoIP	–	Voice over Internet Protocol
W3C	–	World Wide Web Consortium
WebRTC	–	Web Real-Time Communications
XHR	–	XMLHttpRequest
XML	–	Extensible Markup Language

Seznam obrázků

1 Úvod

Real-time komunikace přes Internet je možná od doby kdy se kvalita IP sítí z hlediska zpoždění dostala na dostatečnou úroveň. Od té doby můžeme volat ať už fyzickým IP telefonem, softphonem, nebo přes aplikace jako Skype, Teamspeak, apod. skrze IP sítě. Pořád ale byla nutnost vlastnit dedikovanou aplikaci, popřípadě plugin. To se pokusila změnit společnost Google v roce 2011, kdy vytvořila první implementaci WebRTC. WebRTC je komunikační model pro peer-to-peer přenášení hlasu, videa, ale i dalších real-time médií. Čím se liší od dříve zmíněných způsobů IP real-time komunikace je, že funguje v samotném prohlížeči, bez nutnosti instalace jakýchkoli přídavných modulů, pouze prostřednictvím webové aplikace. Uživatelské rozhraní, například textová pole pro zadávání údajů nebo funkční tlačítka, je realizováno skrze HTML5, ale samotnou komunikaci už poté zajišťuje JavaScript aplikace. Od představení WebRTC uběhlo několik nezbytných let, aby se připravilo pro širší využití. Skupiny IETF a W3C, které vytvářejí internetové standardy, a mimo Google i jiné velké společnosti jako Cisco a Mozilla přispěly ke zdokonalení WebRTC, kooperujících protokolů a kodeků. Výsledkem je fakt, že dnes jej podporují všechny populární prohlížeče, je využitelný v řadě různých implementací, a využívají jej služby celosvětového měřítka jako Facebook Messenger, Google Hangouts, Discord, Amazon Chime, a další. [1]

Pro komunikaci skrze IP sítě obecně potřebujeme signalizaci, která nám domluví parametry přenosu médií a vymění si informace nezbytné pro provoz dané služby. Model WebRTC signalizační protokol nedefinuje. Dříve zmíněné komerční aplikace ovšem nejsou využitelné mimo danou platformu, tudíž mezi nima ani neexistuje interoperabilita a nemohou být integrovány do stávající VoIP infrastruktury. Toto je možné skrze signalizační protokol SIP (Session Initation Protocol), který je současně nejrozšířeněji využívaným standardizovaným signalizačním protokolem ve VoIP. Proto se nachází příhodná možnost implementovat WebRTC do již existující SIP sítě, a při implementaci SIP serveru současně jako WebRTC brány je možné dokonce komunikace s běžnými SIP klienty. WebRTC není schopno posílat SIP zprávy stejně jako klasický SIP klient. Jelikož má prohlížeč vyšší požadavky na zabezpečení, je nucen posílat signalizaci skrze WebSockety a SIP server musí být nakonfigurován tak aby je uměl zpracovat. S touto problematikou souvisí třeba i to, že byly vytvořeny zásady pro využívání WebSocketů jako transportního protokolu pro SIP zprávy.

V první kapitole bude popsán model WebRTC a jeho jednotlivá rozhraní. Následně bude stručně popsáno, jak jsou jednotlivé protokoly zasazeny do WebRTC modelu, a zvláštní pozornost bude věnována signalizačnímu protokolu SIP. Třetí kapitola bude zahrnovat WebSocket standard a protokol, a v poslední teoretické kapitole bude popsán SIP proxy server Kamailio. Cílem praktické části je navrhnout topologii a zprovoznit server Kamailio tak, aby přes něj bylo možné zahájit hovor mezi WebRTC klienty. Jakymile bude topologie provozuschopná, bude cílem zachytit a zobrazit komunikaci mezi těmito entitami.

2 WebRTC

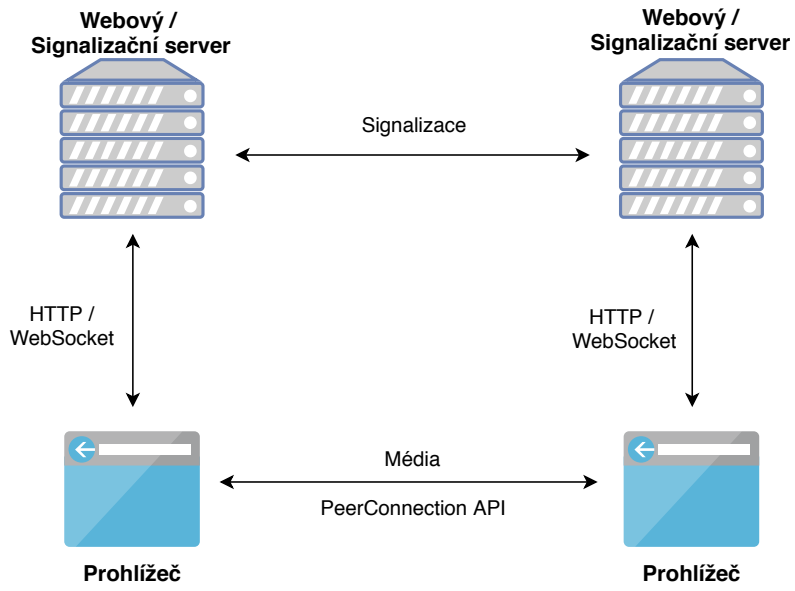
Dříve byla multimediální komunikace v prohlžečích zajišťována pomocí pluginů, jako třeba Adobe Flash nebo Microsoft Silverlight. V roce 2011 uvedla společnost Google první implementaci WebRTC (Web Real-Time Communication), které využívalo přímo HTML5, které bylo schopno přehrávat audio i video bez podpory třetí strany. V této době WebRTC již podporuje většina využívaných prohlížečů. Na dalším vývoji se společně podílí společnosti IETF (RTCWEB Working Group) a W3C (RTC Working Group a Media Capture Task Force). IETF se zaměřuje zejména na standardizaci protokolů, které se využívají pro přenos (bits on wire), ovšem v tomto případě se nesnaží vytvářet nové protokoly, ale dodatky (extensions) k již zavedeným protokolům. Naproti tomu W3C se zaměřuje na vývoj JavaScript API využívané HTML5. [2]

2.1 Webová architektura

Klasická webová architektura je založena na modelu klient-server, kdy prohlížeče posílají HTTP požadavky na obsah web serveru, který posílá odpovědi obsahující požadované informace. Zdroje poskytované serverem jsou blízce spojené s entitou známou skrze identifikátory URI nebo URL. Ve scénáři, kdy máme webovou aplikaci, server může vložit JavaScript kód do HTML stránky, kterou posílá klientovi. Takový kód může spolupracovat s prohlížečem přes standardní JavaScript API a s uživatelem přes uživatelské rozhrání.

2.2 Architektura WebRTC

WebRTC rozšiřuje model klient-server prvkem peer-to-peer komunikace mezi prohlížeči. Většina WebRTC komunikačních modelů vychází z takzvaného SIP Trapezoidu 7.

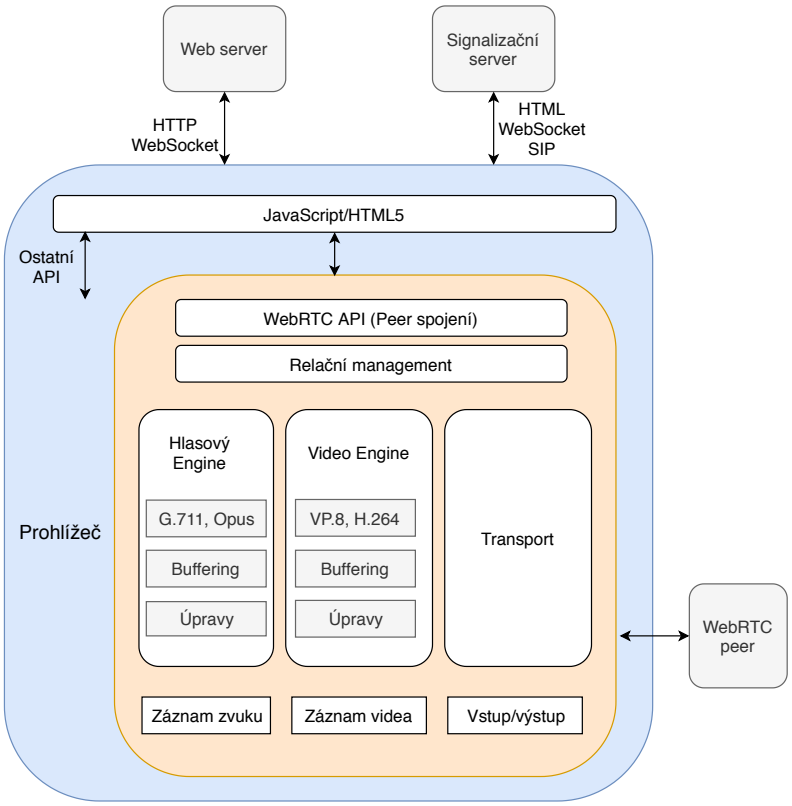


Obrázek 1: WebRTC trapezoid

Oba prohlížeče v WebRTC trapezoidu provozují webovou aplikaci, která je stažena z webového serveru. Signalizační zprávy jsou posílány HTTP nebo WebSocket protokolem přes webové servery, které je umí modifikovat, přeložit nebo manipulovat dle potřeby. V jiném případě signalizace nebude posílána přes webové servery, ale signalizační servery budou samostatné entity. Signalizační protokol jako takový není specifikován WebRTC standardem, ale je považován jako součást aplikace. Samotný hovor je po navázání komunikace mezi klienty veden po nejkratší možné trase. Webové servery mohou komunikovat standardním signalizačním protokolem jako je SIP nebo Jingle (XEP-0166). V opačném případě používají proprietární signalizační protokol. Nejběžnější WebRTC scénář je když oba prohlížeče využívají stejnou webovou aplikaci staženou ze stejného webového serveru, v tomto případě se z trapezoidu stává trojúhelník.

WebRTC webová aplikace (typicky psaná kombinací HTML a Javascript) spolupracuje sCwebovými prohlížeči přes standardizovanou WebRTC API, umožňující správně využívat a řídit realtime funkci prohlížeče.

WebRTC API musí poskytovat širokou škálu funkcí, jako je management spojení, dojednání kódování/dekódování, selekce a řízení, řízení médií, firewall, NAT přechody, atd.



Obrázek 2: WebRTC architektura

2.3 Signalizace

Hlavní představa za designem WebRTC byla plně specifikovat řízení přenosu médií, zatímco signalizaci nechat co možná nejvíc na aplikační vrstvě. Důvod za tímto paradigmatem je ten, že různé aplikace mohou upřednostňovat různé signalizační protokoly.

Popis relace (session description) představuje nejdůležitější informace, které musí být vyměněny. Specifikuje transportní (a ICE) informace, typ a formát médií, a všechny další konfigurační parametry potřebné k sestavení spojení. Typický protokol pro výměnu těchto informací je SDP, který má ovšem pro potřeby WebRTC několik nedostatků. Z toho důvodu se IETF rozhodlo standardizovat protokol JSEP, který aplikaci poskytuje potřebné rozhraní, nehledě na signalizační protokol.

JSEP kompletně přenáší odpovědnost za sinalizační procesy na aplikaci. Aplikace musí oslovit danou API v daný čas a převést popisy relací a příslušnou ICE informaci do zprávy daného signalizačního protokolu.

2.4 WebRTC API

W3C WebRTC 1.0 API umožňuje JavaScript aplikaci využít real-time vlastností nových prohlížečů. Real-time funkce prohlížeče implementovaná v jeho jádru poskytuje funkcionalitu potřebnou k sestavení hlasových (audio), obrazových (video) a dataových kanálů. Všechna média a data jsou zakódovaná s využitím DTLS. Pro zajištění kompatibility mezi různými implementacemi real-time funkcí v prohlížečích IETF zapracovala na výběru minimální povinné sady audio a video kodeků. Opus (RFC6716) a G.711 byly vybrány jako povinné mezi audio kodeky a mezi video kodeky to jsou H264 a VP8. Prohlížeče běžně podporují i video kodek VP9. (RFC 7874/7742) [3][4] WebRTC API je navrženo dle tří hlavních konceptů: `MediaStream`, `PeerConnection`, a `DataChannel`. [5]

2.4.1 MediaStream

`MediaStream` je abstraktní reprezentace proudu audia a/nebo videa. Slouží jako prostředek k řízení proudu médií, jako je zobrazení obsahu, nahrávání, nebo posílání vzdálenému učastníkovi. `MediaStream` může být také reprezentovat proud, který je buď přijímán (remote stream), nebo je posílán (local stream) na vzdálený bod.

`LocalMediaStream` reprezentuje proud médií z lokálního nahrávajícího zařízení (webkamera, mikrofon, apod.). Pro vytvoření a užívání lokálního proudu, web aplikace musí vyslat požadavek pro zpřístupnění uživateli skrze funkci `getUserMedia()`. Aplikace specifikuje typ médií–audio nebo video–ke kterému vyžaduje přístup. Výběr zařízení v prohlížečovém rozhraní slouží jako mechanismus pro udělování nebo zamezování přístupu. Jakmile aplikace skončí se svou prací, může svůj přístup zrušit zavoláním funkce `stop()` na `LocalMediaStream`.

Protokol SRTP je užíván pro přenos médií společně s informacemi definovanými protokolem RTCP, které jsou využívány ke sledování přenosových statistik týkajících se proudů dat. DTLS se používá pro pro SRTP klíčování a řízení sdružování. [6] [7]

V multimediální komunikaci bývalo každé médium typicky přenášeno v oddělené RTP relaci se svými vlastními RTCP pakety. To znamená, že každá relace si vytvořila svůj vlastní NAT překlad (neefektivní vužívání portů), prodlužovala se doba navázání spojení a zvyšovalo se riziko selhání. Na tomto IETF zapracovalo skupinou dodatků (RFC 8108). [8] [9]

2.4.2 PeerConnection

`PeerConnection` umožňuje dvěma uživatelům komunikovat přímo z prohlížeče na prohlížeč. Reprezentuje asociaci se vzdáleným účastníkem, který je obvykle jiná instance stejné JavaScript aplikace provozována na vzdáleném konci. Komunikace je koordinována přes signalizační kanál. Jakmile je navázáno spojení účastníků, proudy médií (lokálně definovány `MediaStream` objekty) mohou být posílány napřímo vzdálenému prohlížeči.

`PeerConnection` mechanismus využívá ICE protokol společně se STUN a TURN servery k posílání UDP proudů médií přes NAT brány a firewally. ICE umožňuje prohlížečům získat dostatek informací o topologii sítě, ve které jsou nasazeny, k nalezení nejlepší použitelné cesty. Použití ICE taktéž poskytuje bezpečnostní opatření, jelikož znemožňuje nedůvěryhodným webovým stránkám a aplikacím posílat nežádoucí data účastníkům komunikace. [10] [11] [12]

2.4.3 DataChannel

`DataChannel` API je navrženo k poskytování obecných transportních služeb, které umožňují webovým prohlížečům vyměňovat si mezi sebou data v obousměrné peer-to-peer komunikaci. Standartizační práce IETF dospěla k rozhodnutí používat SCTP enkapsulováno pomocí DTLS pro ne-mediální datové typy.

Enkapsulace SCTP přes DTLS přes UDP spolu s ICE poskytuje řešení NAT přechodů, dále také důvěryhodnost, strojovou autentikaci a integritu zabezpečeného přenosu. Navíc toto řešení umožňuje přenosům dat bezproblémově spolupracovat s přenosy médií a potencionálně mohou všechny tyto přenosy sdílet jeden port. SCTP byl zvolen z důvodu nativní podpory více proudů dat v buď spolehlivém nebo částečně spolehlivém módu. Poskytuje možnost vytvoření několika nezávislých proudů uvnitř SCTP svazku. Každý proud představuje jednosměrný kanál poskytující "iluzi"sekvenční komunikace. Sekvence zpráv může být buď v pořadí nebo mimo pořadí. Pořadí zpráv je zachováno pouze pro zprávy posílaných ve stejném proudu. Nicméně `DataChannel` API je navržen obousměrně, což znamená, že se `DataChannel` skládá z příchozího a odchozího SCTP proudu.

`DataChannel` nastavení je vykonáno prvním zavoláním funkce `CreateDataChannel()` na instanci objektu `PeerConnection`. Každé následující volání funkce `CreateDataChannel()` vytvoří pouze nový `DataChannel` v již existujícím SCTP svazku. [13]

3 Protokoly ve WebRTC

Jak již bylo dříve zmíněno, vývoj protokolů využívaných WebRTC má na starost skupina IETF. Jedná se zejména o vhodnou implementaci již existujících protokolů a právě pro implementace protokolů ve WebRTC byly vytvořeny nové RFC, popř. drafty. Těmito protokoly se snaží o zajištění funkcí potřebných pro telefonní nebo videokonferenční aplikace v prohlížeči.

Funkcionality vyžadované v prohlížeči mohou být rozděleny do 6 funkcionálních skupin:

- **Přenos dat** - TCP/UDP a prostředky pro bezpečné navázání spojení, funkce pro rozhodování, kdy posílat data, atd. Koncové body musí implementovat transportní protokoly momentálně popsané v draftu [14].
- **Rámcování a zabezpečování** - RTP, SCTP, DTLS, a další formáty, které slouží jako kontejnery, a jejich funkce pro věrohodnost a integritu dat. Pro RTP je vyžadována implementace SRTP.[8] [6] [7]
- **Formáty dat** - specifikace kodeků, formátů, a funkcionalit pro data přenášené mezi systémy. Do této kadegorie spadají zejména audio a video kodeky.
- **Relační management** - zahajování spojení, domluva na formátech dat, změna datových formátů během hovoru; do této kategorie spadají: SDP, SIP, Jingle/XMPP.
- **Prezence a řízení** - toto je zejména lokální funkce mezi prohlížečem, operačním systémem a uživatelským rozhráním.
- **Podpůrné funkce lokálního systému** - zahrnují eliminaci echa, kontrolu hlasitosti, řízení kamery včetně zaostřování, apod.

První 3 funkcionální skupiny tvoří infrastrukturu přenosu médií, další 3 skupiny tvoří mediální služby a alespoň prvních 5 skupin musí být specifikováno. Používané protokoly jsou hiearchicky zobrazeny v obrázku 3 . [15]

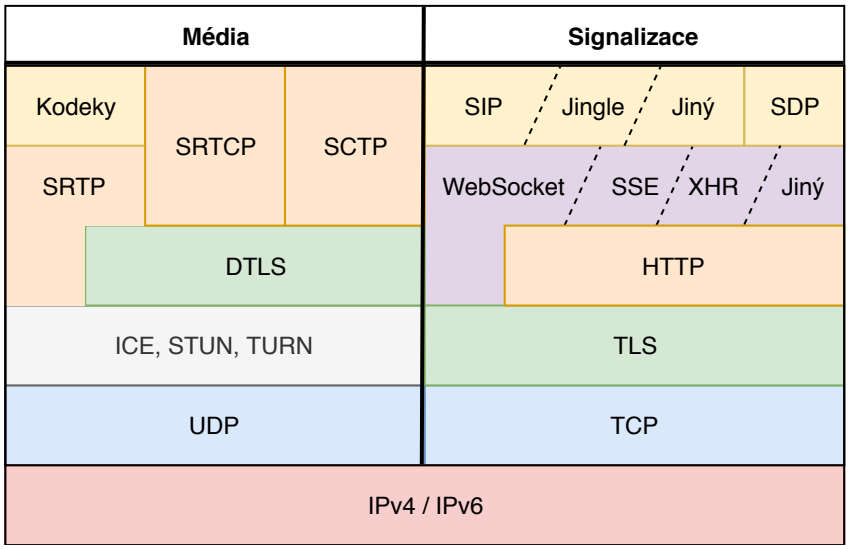
3.1 SIP

Je to textově založený řídící protokol aplikační vrstvy, který byl navrhnut a vytvořen skupinou IETF a kategorií spadá pod VoIP protokoly. Nejdůležitější RFC dokument pro SIP je RFC 3261, který specifikuje jádro protokolu. [16]

Za zmínku stojí i to, že SIP je založený na protokolu HTTP, tudíž lze mezi nima nalézt podobnost v hlavičkách a kódových označeních odpovědí.

3.1.1 Význam SIP

Jak již napovídá název (Session Initation Protocol), jde o protokol, jehož účelem je vytváření, modifikace a ukončování relace s jedním či více účastníky. Relací rozumíme soubor odesílatelů,



Obrázek 3: WebRTC protokoly

příjemců a stav udržovaný mezi těmito odesílateli a příjemci během jejich komunikace. Příkladem takových relací mohou být internetové telefonní hovory, konference, streamování apod. Jsou zde ale i další protokoly, které se na komunikaci podílí. SIP typicky spolupracuje s protokoly RTP a SDP. RTP se používá pro přenos médií, stará se o kódování a dělení dat do paketů. Jeho zabezpečená verze je SRTP. SDP je textově založený protokol sloužící pro popis a vyjednání parametrů spojení, jako např. použité kodeky a typy médií. [17]

SIP podporuje 5 základních aspěktů vytváření a ukončování komunikace:

- **Lokace uživatele** - určení koncového systému, který pro komunikaci bude používat
- **Dostupnost uživatele** - určení, zdali je volaná strana ochotna komunikovat
- **Schopnosti uživatele** - určení médií a jejich parametrů, které se budou používat
- **Nastavení relace** - vyzvánění (ringing) a navázání relačních parametrů na volané i volající straně
- **Relační management** - přenos a ukončení relací, modifikace jejich parametrů a vyvolávání služeb

[16]

3.1.2 SIP URI

SIP účastníci jsou indentifikováni za pomoci SIP URI. To má následující formu: `sip:jméno@doména`, například `sip:bob@vsb.cz`. Jak můžeme vidět, skládá se z uživatelského jména, pod kterým je

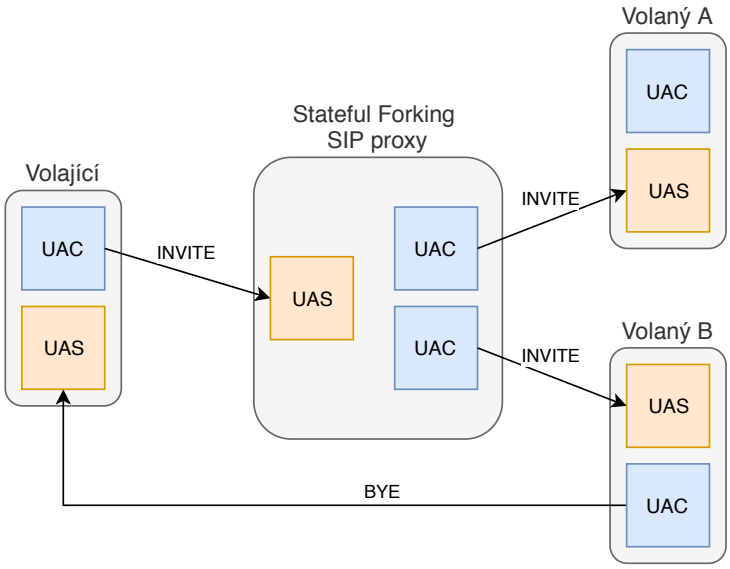
účasník zaregistrován, a doménového jména serveru, na kterém je uživatelské jméno zaregistrováno. Velmi podobně jsou řešeny emailové adresy. Doménové jméno lze zaměnit za IP adresu serveru a celé SIP URI lze zaměnit za IP adresu adresáta, ovšem SIP URI je ze své podstaty snáze zapamatovatelné.

3.1.3 Prvky SIP architektury

Přestože by si v jednoduché topologii vystačili dva účastníci sami, běžná topologie se skládá z více SIP entit. Základní z nich jsou user agenti, proxy, registrar a redirect servery. Tyto entity jsou převážně logického charakteru, jedno zařízení často zastává více těchto entit, ale záleží na konkrétní implementaci.

User agent Koncové body telekomunikační sítě, které jako signalizační protokol pro sestavení spojení používají SIP, nazýváme *User Agent* (UA). V dnešní době může UA představovat široká škála zařízení a programů. User agent je často označován i jako *User Agent Server* (UAS) a *User Agent Client* (UAC). UAS i UAC jsou ovšem pouze logickými entitami, každý User agent je zároveň UAC i UAS.

UAC je část user agenta, která posílá žádosti a přijímá odpovědi, oproti tomu UAS je část, která přijímá žádosti a posílá odpovědi, viz obr. 4 . Jelikož user agent představuje zároveň UAS i UAC, podle situace o něm říkáme, že se chová jako UAS nebo UAC.



Obrázek 4: UAC a UAS

B2BUA je speciální typ user agenta, který zastává funkci SIP proxy serveru, ale chová se jako UA. Typickým příkladem je **Asterisk**. Oproti SIP proxy může poskytovat řadu pokročilejších doplňkových služeb a proto je využíván jako pobočnová ústředna pro řádově až tisíce uživatelů. Oproti tomu SIP proxy je využitelná až pro statisíce uživatelů.

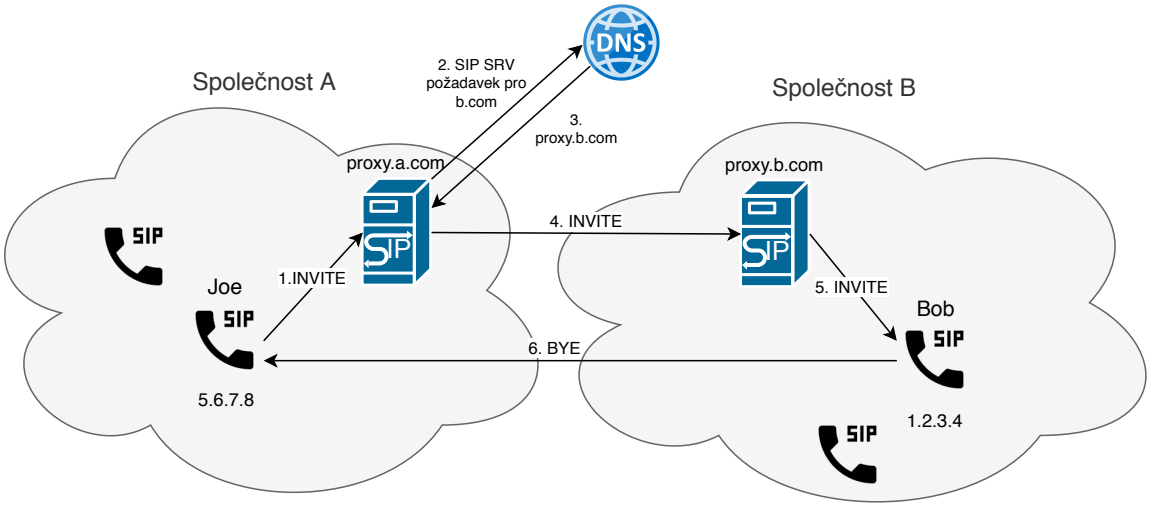
SIP proxy Jsou to velmi důležité prvky SIP infrastuktury, které primárně směrují SIP zprávy, ale mohou poskytovat i mnoho dalších služeb jako registraci, autentizaci atd. Typickým příkladem je **Kamailio**. Vzhledem ke konfiguraci rozlišujeme 2 typy SIP proxy serveru – stateful a stateless (stavové a bezestavové).

- **Stateless** Tyto SIP proxy pracují pouze jako jednoduché směrovače, které SIP zprávy směrují nezávisle na předchozích. Přestože jsou tyto zprávy obvykle uspořádány do transakcí, stateless proxy na to nebere ohled. Díky tomu jsou řádově rychlejší než stateful proxy, ale na druhou stranu nejsou schopny vykonávat pokročilejší operace jako větvení nebo přesměrování, nezachytí replikaci zpráv a detekce nekonečných smyček jim trvá déle.
- **Stateful** Tyto SIP proxy jsou komplexnější. Po přijetí požadavku server vytvoří záznam, který uchovává až do ukončení transakce nebo dialogu, dle toho se dělí na **transakční** a **dialogové**. Některé transakce mohou trvat poměrně dlouho, obzvláště ty vytvořené metodou INVITE, což várazně snižuje výkon serveru. Schopnost asociovat si jednotlivé zprávy do transakcí dává stateful proxy zajímavé vlastnosti.
  - **Větvení** - na základě přijetí jedné zprávy server může server vyslat 2 a více zpráv, viz. obr 4 .
  - **Zachycení opakovaných zpráv** - z uchovávaného záznamu může zjistit, jestli už byla stejná zpráva přijata.
  - **Nalezení uživatele** - může provádět komplikovanější metody pro nalezení uživatele, například v situaci ve které volaný nezvedá telefon v kanceláři, je hovor přesměrován na jeho mobilní telefon.

  Většina dnešních SIP proxy jsou stateful a často podporují generování záznamů o spojeních, větvení a některé typy NAT.

Bežně má každá centrálně administrovaná entita, například firma, vlastní SIP proxy server, využívaný všemi UA v dané entitě. Předpokládejme, že máme společnosti A a B a obě mají svůj vlastní proxy server. Obrázek 5 ukazuje jak se zpráva pro zahájení relace od zaměstnance Joe z firmy A dostane k zaměstnanci Bob z firmy B. Joe použije URI adresu `sip:bob@b.com`. Joeův UA je nakonfigurován tak, že všechny výchozí žádosti posílá na `proxy.a.com` (zná IP adresu). Tento proxy server z URI adresy `sip:bob@b.com` pozná, že se volaný nachází v jiné firmě, proto pokud nemá v paměti patričný záznam, přeloží si přes DNS server adresu `proxy.b.com` na příslušnou IP adresu, kam posléze zprávu přepošle. Proxy server `proxy.b.com` ví, že Bob právě sedí ve své kanceláři a je dostupný přes telefon na jeho stole, který má IP adresu `1.2.3.4`, na kterou zprávu pošle. Odpověd již není třeba posílat přes SIP proxy, jelikož v žádosti byla obsažena fyzická adresa odesílatele.

Registrar V předchozím příkladu bylo zmíněno, že SIP proxy server `proxy.b.com` sice zná Bobovu aktuální polohu, ale aby měl tuto informaci, Bobův UA musí být registrován na SIP



Obrázek 5: SIP topologie

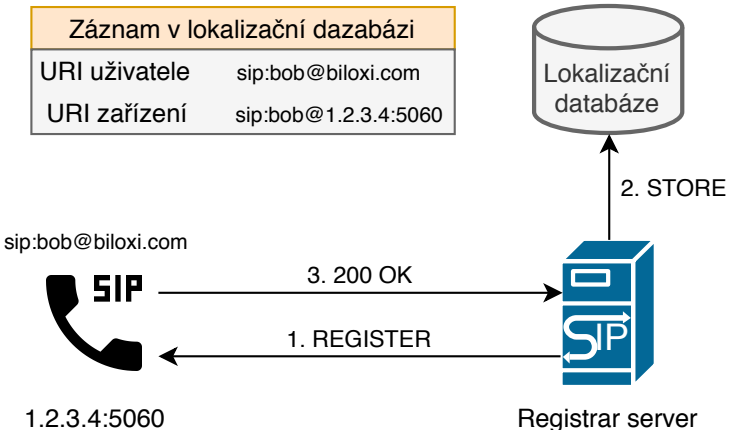
registrar serveru. Registrar server je speciální entita, která od uživatelů přijímá registrace, tím získává informace o jejich aktuální poloze (v tomto případě IP adresa, port a uživatelské jméno) a tyto informace si ukládá do lokalizační databáze (location database). Účel lokalizační databáze je v tomhle případě přeložit adresu `sip:bob@b.com` na `sip:bob@1.2.3.4:5060`. Obrázek 6 znázorňuje typickou SIP registraci. UA na registrar pošle zprávu metody REGISTER, která obsahuje adresu záznamu User URI `sip:bob@biloxi.com` a Device URI `sip:bob@1.2.3.4:5060`, kde adresa `1.2.3.4` je IP adresa telefonu. Registrar tuto informaci přečte a uloží ji do lokalizační databáze. Pokud všechno proběhlo vpořádku, registrar pošle telefonu odpověď 200 OK a proces registrace je hotov.

Každá registrace má omezenou životnost, hlavička `Expires` nebo parametr `expires` v hlavičce `Contact` udává, po jakou dobu je registrace platná. UA proto musí registraci obnovovat, v opačném případě platnost registrace vyprší.

[17]

Redirect server Entita, která přijímá požadavky a posílá zpět odpovědi obsahující seznam současných poloh daného uživatele je nazývána *redirect server*. Redirect server přijme požadavek a vyhledá si daného adresáta v příslušné lokalizační databázi vytvořené registrar serverem. Poté vytvoří seznam současných lokací a pošle jej zpět tvůrci žádosti v odpovědi třídy 3xx. Toto může být užitečné například v situaci, kdy se osoba zrovna nachází v doméně `cvut.cz` a nabízí se jí možnost využívat jejich SIP proxy. Běžně ovšem využívá SIP proxy v doméně `vsb.cz`, takže by ji bez redirect serveru nebylo možné kontaktovat pomocí původní SIP URI.

[18]



Obrázek 6: SIP registrace

3.1.4 SIP zprávy

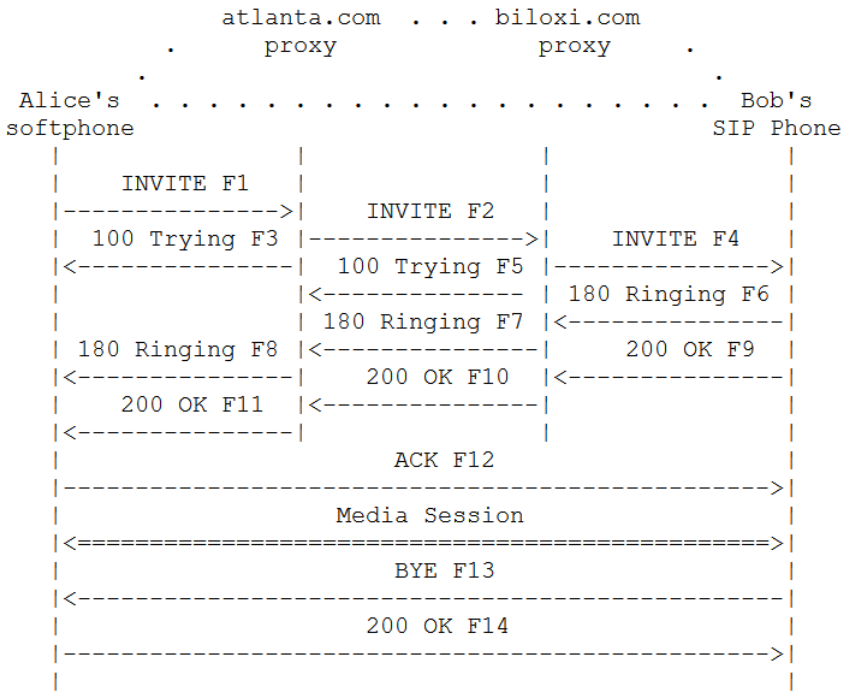
SIP signalizace se skládá ze série zpráv, které jsou buď žádosti nebo odpovědi, a tyto zprávy jsou nezávisle posílány přes danou síť. Typicky jsou neseny v jednotlivých UDP datagramech, ačkoli je možné použít i TCP protokol. Každá SIP zpráva se skládá z "prvního řádku", hlavičky a těla, které je definováno SDP protokolem. Přenos těchto SIP zpráv mezi jednotlivými SIP prvky znázorňuje tzv. SIP trapezoid, viz. obr. 7

Struktura zprávy je následující:

- **První řádek** (start-line) - liší se u požadavku a odpovědi
  - **Řádek požadavku** (request-line) - při požadavku
    - **Metoda**
    - **SIP URI**
    - **Verze SIP** - typicky 2.0
  - **Řádek statusu** (status-line) - při odpovědi
    - **Verze SIP**
    - **Statusový kód** (status-code) - třímístné číslo značící povahu odpovědi
    - **Důvod** (reason-phrase) - oproti statusovému kódu je vyjádřen textem a určen pro lidského účastníka
- **Hlavička** (message-header)
  - **Via** - každé zařízení v rámci požadavku připíše svůj "Via"řádek, do kterého zapíše svou fyzickou adresu a identifikátor transakce (branch), u odpovědi je zpráva posílána stejnou cestou zpět a tyto řádky se odebírají
  - **Max-Forwards** - udává kolik skoků může zpráva do cíle provést (mezi SIP síťovými prvky)

- **To** - obsahuje zobrazované jméno (display name) a SIP URI, na kterou je zpráva směřována
- **From** - obsahuje zobrazované jméno a SIP URI odesílatele zprávy, taktéž obsahuje parametr tag obsahující náhodnou sérii znaků pro identifikační účely
- **Call-ID** - obsahuje globálně unikátní identifikátor pro daný hovor generovaný náhodnou kombinací znaků a jménem koncového zařízení nebo IP adresou; kombinace parametrů To tag, From tag, a Call-ID kompletně definuje peer-to-peer vztah mezi účastníky a je označován jako dialog
- **Cseq** (Command sequence) - obsahuje číslo, které je inkremetováno každým dalším požadavkem, a název metody
- **Contact** - obsahuje SIP URI, které reprezentuje přímou cestu pro kontaktování odesílatele, bežně složené z plně specifikovaného doménového jména (FQDN), ale je možno použít i IP adresu
- **Content-Type** - uvádí jaké ho typu (povahy) je tělo zprávy
- **Content-Lenght** - uvádí, kolik bytů má tělo zprávy

- **Prázdný řádek** (CRLF)
- **Tělo** (message-body) - SDP



Obrázek 7: SIP trapezoid

Příklad SIP zprávy F1 v trapezoidu 7 :

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

(Zde následuje SDP)

[16]

4 WebSocket

WebSocket je soustava několika standardů: WebSocket API je definováno W3C, naproti tomu WebSocket protokol (RFC 6455) je definován skupinou HyBi Working group (IETF). Umožňuje obousměrný, zprávově orientovaný přenos textu a binárních dat mezi klientem a serverem. Websocket API je rozhraní prohlížeče nejblíže k síti a v rámci síťového spojení poskytuje několik dodatěčných služeb:

- Dohodnutí spojení a ochranu původu (same-origin policy)
- Interoperabililta se současnou HTTP infrastrukturou
- Zprávově orientovaná komunikace a efektivní rámcování
- Vyjednání subprotokolu a rozšiřitelnost

WebSocket je nejvšestranější a nejflexibilnější transportní technologie dostupná v prohlížečích. Jednoduchá API dovoluje posílat libovolná data z libovolné aplikace mezi klientem a serverem – cokoliv od jednoduchých JSON payloadů po libovolný formát binárních zpráv – na principu streamů, kdy kterákoliv strana může poslat data v jakýkoli moment.

Nicméně jsou faktory, které je třeba zvážit. Každá aplikace musí počítat s absencí řízení, komprese, cache a dalšími službami jinak poskytovanými prohlížečem. WebSocket není náhrada za HTTP, XHR nebo SSE a pro nejlepší výkon je zásadní zvážit klady a zápory každého transportního protokolu.

4.1 WebSocket API

Websocket API poskytované prohlížečem je znatelně malé a jednoduché. Všechny nízkoúrovňové detaily řízení a zpracování zpráv jsou zajištěny samotným prohlížečem. K vytvoření nového spojení potřebujeme URL WebSocket zdroje (resource) a několik callbacků (funkce která se vykoná jako odezva na nějakou událost):

```
var ws = new WebSocket('wss://example.com/socket'); //1

ws.onerror = function (error) { ... } //2
ws.onclose = function () { ... } //3

ws.onopen = function () { //4
  ws.send("Connection established. Hello server!"); //5
}

ws.onmessage = function(msg) { //6
```

```
  if(msg.data instanceof Blob) { //7
    processBlob(msg.data);
  } else {
    processText(msg.data);
  }
}
```

1. Otevření nového zabezpečeného spojení (wss)
2. Volitelný callback, vyvolaný chybou spojení
3. Volitelný callback, vyvolaný pokud je spojení přerušeno
4. Volitelný callback, vyvolaný návázáním WebSocket spojení
5. Klientem vyvolaná zpráva poslaná serveru
6. Callback funkce vyvolaná pro každou novou zprávu ze serveru
7. Vyvolaný binární, nebo text zpracovávající proces pro obdrženou zprávu

4.1.1 WebSocket URL

Websocket URL používá své vlastní schéma: `ws` pro čistě textovou komunikaci a `wss` pro zabezpečený kanál (TCP+TLS) Například `ws://example.com/socket`. Vlastní schéma nebylo navrženo bezdůvodně. WebSocket lze použít i mimo prohlížeč a bez vazby na HTTP.

4.1.2 Přijímání textu a dat

WebSocket komunikace se skládá ze zpráv a kódu aplikace, nemusí se starat o vyrovnávací paměť, parsování a rekonstrukci dat. Například, pokud server pošle 1 MB payload, callback aplikace `onmessage` bude zavolán pouze pokud klient obdrží celou zprávu. Dále WebSocket protokol nevytváří žádné předpoklady a neomezuje payload na aplikaci – s textem i binárními daty pracuje stejně. Protokol na zprávě sleduje jenom 2 proměnné: délku paylloadu a jeho typ (UTF-8 nebo binární data).

Když je nová zpráva přijata prohlížečem, automaticky je převedena na objekt `DOM-String` pro text nebo `Blob` pro binární data a posléze předána aplikaci. Další možnost je převést přijatá binární data na objekt `ArrayBuffer`, což může sloužit pro optimalizaci výkonu v případě, že budou data ponechány v paměti. V případě uložení dat na disk je vhodnější použití objektu `Blob`.

4.1.3 Posílání textu a dat

Jakmile je WebSocket spojení navázáno, klient může posílat a přijímat UTF-8 a binární zprávy dle libosti. WebSocket API přijímá objekty typu DOMString, ArrayBuffer, ArrayBufferView, nebo Blob, jak bylo zmíněno výše. Tyto objekty jsou ovšem API přizpůsobení: WebSocket API má pro typ dat vyhrazen 1 bit. Z toho vyplývá, že pokud některá z komunikujících stran potřebuje dodatečné (kontextové) informace o přenášených datech, musí k jejich získání použít dodatečný mechanismus.

Metoda `send()` je asynchronní: poskytovaná data jsou klientem řazena do fronty, a funkce se vrací okamžitě. Tudíž, obzvláště při posílání objemnějších payloadů, se nesmí zaměnit vykonání funkce a reálný přenos dat. K monitorování množství dat ve frontě prohlížeče se aplikace může dotázat atributem `bufferedAmount` na socketu.

```
var ws = new WebSocket('wss://example.com/socket');

ws.onopen = function () {
  subscribeToApplicationUpdates(function(evt) { //1
    if (ws.bufferedAmount == 0) //2
      ws.send(evt.data); //3
  });
};
```

1. Přihlášení se k odebírání aktualizací
2. Ověření fronty dat u klienta
3. Vyslání dalších dat, pokud je fronta prázdná

V tomto příkladě se pokoušíme o posílání zpráv na server, ale pouze pokud předchozí zprávy již opustily frontu klienta. Proč se ale obtěžovat s takovou podmínkou? Všechny WebSocket zprávy jsou doručovány v takovém pořadí v jakém jsou řazeny klientem. Ve výsledku by velká fronta nevyřízených zpráv, nebo i jedna velká zpráva vytvářela zpoždění pro všechny zprávy za ní (head-of-line blokování). Aplikace se s tímto problémem vypořádává rozdělováním velkých zpráv na menší části, monituruje hodnotu `bufferedAmount` a implementuje vlastní prioritní frontu pro nevyřízené zprávy.

4.1.4 Vyjednávání subprotokolu

Kromě již dříve zmíněného, WebSocket protokol nerozlišuje formát zpráv: jedním bitem se zjišťuje, jestli zpráva obsahuje text nebo binární data, tak aby mohla být efektivně dekódována clientem a serverem, ale jinak je obsah zprávy neznámý.

Narozdíl od HTTP nebo XHR požadavků, které dodatečné metadata (= data obsahující informace o jiných datech) předávají v HTTP hlavičkách každého požadavku a odpovědi, WebSocket zprávy nemají žádný ekvivalení mechanismus. Pokud jsou dodatečné metadata vyžadovány, klient a server se musí dohodnout na implementaci vlastního subprotokolu pro výměnu těchto dat:

- Klient a server se můžou dohodnout na pevném formátu zprávy předem – například bude všechna komunikace probíhat skrze JSON kódované zprávy nebo volitelný binární formát – a nezbytné metadata budou součástí kódové struktury
- Pokud klient a server potřebují přenést jiné typy dat, mohou se dohodnout na hlavičce zprávy, která bude používána k výměně instrukcí k dekódování zbytku payloadu
- Může být použita kombinace textu a binárních zpráv – například textová zpráva ekvivalentní HTTP hlavičkám následována binární zprávou se zbylým aplikačním payloadem

Tento seznam obsahuje pouze malou část možných postupů. Flexibilita a minimální objem dodatečných informací WebSocket zprávy přichází s cenou dodatečných postupů v aplikaci. Nicméně, serializace zpráv a management metadat jsou pouze část problému. Jakmile ustanovíme serializační proces pro naše zprávy, jak zajistíme, aby si klient a server rozuměli a jak se udržíme v synchronizaci? Naštěstí, WebSocket poskytuje jednoduché a příhodné API pro vyjednání subprotokolu (*subprotocol negotiation API*). Klient může oznámit serveru, jako součást úvodního handshake, které protokoly podporuje

```
var ws = new WebSocket('wss://example.com/socket',
                       ['appProtocol', 'appProtocol-v2']); //1

ws.onopen = function () {
  if (ws.protocol == 'appProtocol-v2') { //2
    ...
  } else {
    ...
  }
}
```

1. Subprotokoly, které klient oznámí během WebSocket handshake
2. Ověření subprotokolů zvolených serverem

V tomto příkladu WebSocket konstruktor přijímá volitelné pole názvů subprotokolů, což klientovi umožňuje vyjádřit, jakým protokolům "rozumí", nebo které chce použít pro dané spojení.

Server je oprávněn si z tohoto seznamu vybrat jeden subprotokol. Pokud je vyjednání subprotokolů úspěšné, na klientovi je vyvolán `onopen` callback a aplikace se může dotázat na atribut `protokol` WebSocket objektu k zjištění zvoleného protokolu. Na druhou stranu, pokud server nepodporuje žádný z oznámených protokolů, WebSocket hanshake nebude dokončen: bude vyvolán `onerror` callback a spojení bude terminováno. [19] [20]

4.2 WebSocket protokol

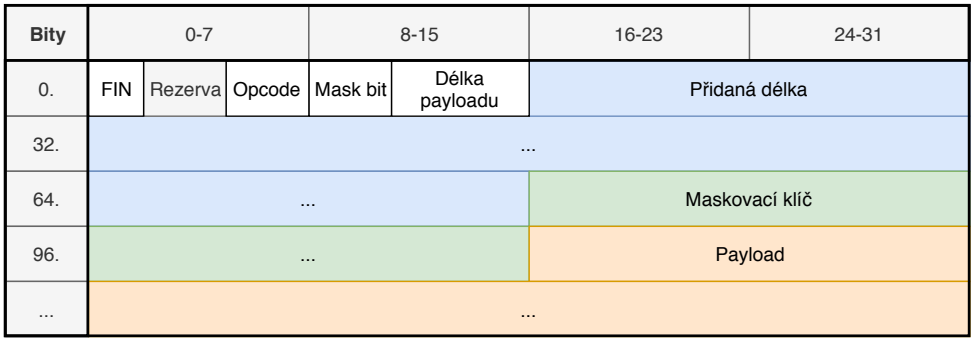
WebSocket (wire) protokol (RFC 6455) se skládá ze dvou částí:

- Úvodní handshake pro vyjednání parametrů spojení - využívá HTTP sémantiku, umožňující Websocket protokolu použít již zavedenou HTTP infrastrukturu
- Mechanismus binárního rámcování pro posílání textu a dat

WebSocket protokol je plně funkční, samostatný protokol, který může být použit i mimo prohlížeč, ale jeho primární využití je jako obousměrný transportní protokol v prohlížečových aplikacích.

4.2.1 Vrstva binárního rámcování

WebSocket aplikace na klientu a serveru komunikují skrze zprávově orientované API: odesílatel poskytuje volitelnou UTF-8 zprávu nebo binární payload, a příjemce je obeznámen, pokud je dostupná celá zpráva. Aby toho bylo docíleno, WebSocket používá vlastní formát rámcování 8, který rozděluje každou zprávu aplikace do jednoho či více rámců, transportuje je do cíle, znovu je sestaví a nakonec oznámí příjemci, jakmile byla obdržena celá zpráva.



Obrázek 8: WebSocket rámec

Rámec Nejmenší jednotka komunikace, obsahující hlavičku proměnné délky a payload, který může nést všechny nebo jednu část zprávy.

Zpráva Kompletní sekvence rámců, která reprezentuje zprávu aplikace.

- První bit každého rámce (FIN) indikuje, jestli je rámec posledním fragmentem zprávy
- 3 bity tvoří rezervu
- Opcode (4 bity) indikuje typ přenášeného rámce: pokračování rámce (0); text (1); binární data (2); rezerva pro ne-řídící rámce (3-7); uzavření spojení (8); ping(9); pong(10); rezerva pro řídící rámce (11-15)
- Mask bit indikuje, jestli je má payload masku (pouze pro zprávy od klienta na server)
- Délka payloadu (7 bitů) je reprezentována polem proměnné délky:
  - 0-125 znamená délku payloadu
  - 126 indikuje, že následující 2 byty reprezentují 16-bitovou kladnou hodnotu, která je délkou rámce
  - 127 indikuje, že následující 2 byty reprezentují 64-bitovou kladnou hodnotu, která je délkou rámce
- Masking key obsahuje 32-bitovou hodnotu používanou k maskování payloadu (zabezpečovací mechanismus)
- Payload obsahuje aplikační data a vlastní dodatečná data, pokud se na nich klient a server dohodli při vytváření spojení

Rozdělování zpráv do rámců je vykonáváno implementací rámcovacího kódu klienta a serveru. Z toho důvodu si aplikace nejsou vědomy individuálních WebSocket rámců nebo jak je rámcování prováděno. [20] [21]

4.3 WebSocket SIP subprotokol

IETF specifikovalo WebSocket SIP subprotokol pro přenos SIP požadavků a odpovědí tak, aby je bylo možné přenést skrze WebSocket spojení. SIP WebSocket klient a SIP WebSocket server se musí dohodnout na užití tohoto subprotokolu zahrnutím hodnoty "sip"v hlavičce "Sec-WebSocket-Protocol"v požadavku úvodního hanshaku. Stejnou hodnotu musí obsahovat i příslušná hlavička odpovědi. Pro jednotlivé SIP zprávy dále platí, že každá musí být nesena v jediné WebSocket zprávě a žádná WebSocket zpráva nemůže nést více než jednu SIP zprávu.

Dále byly pro tuto implementaci definovány některé nové zásady pro protokol SIP.

4.3.1 Neplatná SIP URI

Součástí SIP URI je některých případech typicky IP adresa a port klienta, ovšem SIP WebSocket klienti nebo obecně aplikace v prohlížeči nemají s ohledem na bezpečnost žádný mechanismus

pro zjištění jejich lokální adresy. Proto používají náhodné doménové jméno. Tento nedostatek je řešen na SIP serverech, přes které se u WebSocketového spojení posílá všechna signalizace i po navázání komunikace mezi klienty.

4.3.2 Transportní parametr v SIP URI

Hodnota "ws"parametru "transport"pro SIP URI (RFC3986) značí kontaktování využívající WebSocket subprotokol. Následuje příklad použití takovéto SIP URI v havičce Contact.

```
Contact: <sip:qggau7pf@4t1q4p2m66rn.invalid;transport=ws;ob>
```

4.3.3 Transportní parametr v hlavčce Via

Hlavička Via nese svůj identifikátor použitého transportního protokolu. Ten může obsahovat buď hodnotu "WS"pro základní WebSocketové spojení, nebo hodnotu "WSS"pro šifrované WebSocketové spojení (využívající TLS). Následuje ukázka ze zprávy přijaté na SIP proxy od WebRTC klienta.

```
Via: SIP/2.0/WSS 4t1q4p2m66rn.invalid;branch=z9hG4bK5330440
```

4.3.4 Parametr "received"v hlavičce Via

V RFC3261 je uvedeno, že když server obdrží požadavek, musí zkontrolovat hodnotu parametru "sent-by"(adresa odesílatele) ve vrchní Via hlavičce. Pokud je tato hodnota doménou nebo adresou lišící se od zdrojové adresy paketu, server do této hlavičky musí přidat parametr "received"obsahující zdrojovou adresu paketu. RFC7118 toto tvrzení upravuje tak, že se SIP WebSocket server při přijetí takovéto žádosti může rozhodnout, zdali do ní tento parametr přidá a WebSocket SIP klient musí akceptovat odpovědi bez tohoto parametru ve vrchní hlavičce Via nehledě na to, jestli její parametr "sent-by"obsahuje doménu nebo adresu lišící se od zdrojové adresy paketu. Následuje příklad, jak SIP proxy poupravila hlavičku z předchozí ukázky.

```
Via: SIP/2.0/WSS 4t1q4p2m66rn.invalid;rport=49586;received=192.168.2.219;branch
    =z9hG4bK5330440
```

[22]

5 SIP proxy Kamailio

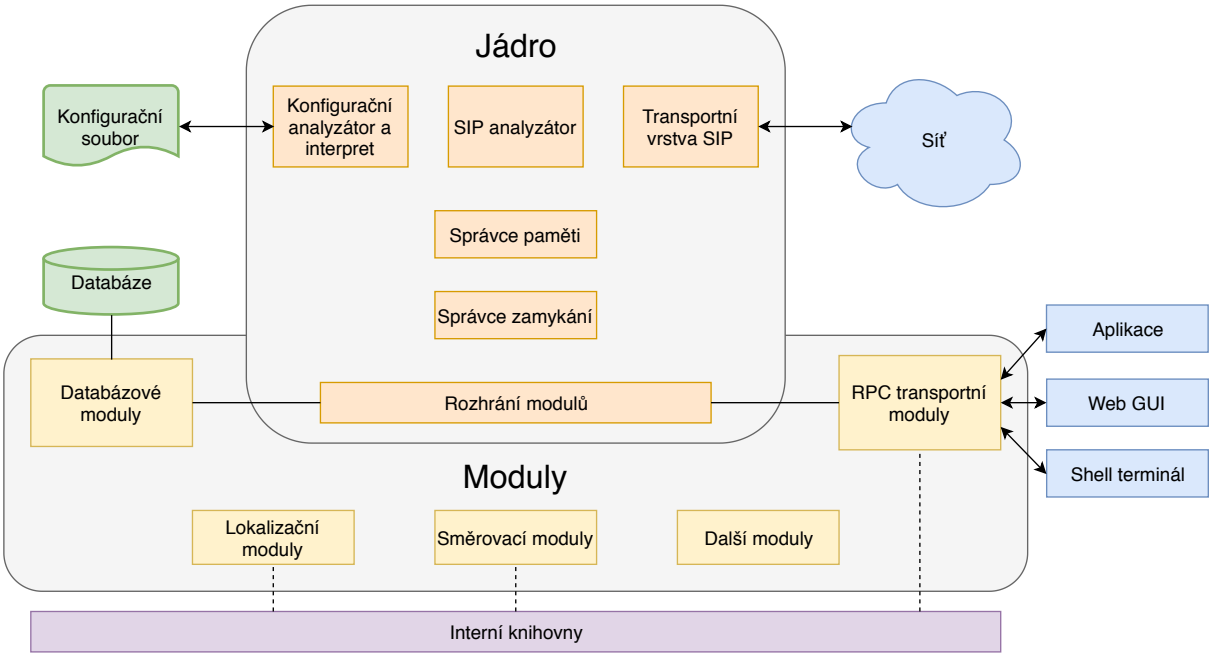
V červnu 2005 Kamailio vzniklo rozdělením projektu SIP Express Routeru (SER). Nově započatý projekt měl za cíl vytvořit open-source vývojové prostředí pro stavbu robustního, škálovatelného open-source SIP serveru. Původní název byl OpenSER, ale z důvodu porušení ochranné známky byl v červenci roku 2008 název projektu změněn na Kamailio. V listopadu roku 2008 se vývojářské týmy Kamailia a SERu znovu spojily a stejně tak konvergovaly i jejich projekty Kamailiem verze 3.0.0. V době psaní této práce je Kamailio ve verzi 5.x.

5.1 Architektura

Kamailio má modulární architekturu. Zjednodušeně jeho komponenty můžeme rozdělit do dvou kategorií:

- **Jádro** - komponenta, která poskytuje nízkoúrovňové funkcionality. Počínajíc verzí 3.0.0, jádro Kamailia obsahuje takzvané interní knihovny. Tyto knihovky sbírají kód sdílený několika moduly, u kterých není důvod, aby byly součástí jádra.
- **Moduly** - komponenty, které poskytují většinu funkcionalit

V obrázku 9 je přibližně znázorněna současná architektura Kamailia.



Obrázek 9: Architektura Kamailia

5.1.1 Jádro

Jádro zahrnuje:

- správce paměti
- analyzátor SIP zpráv
- systém zamykání
- správce DNS a transportní vrstvy (UDP, TCP, TLS, SCTP)
- analyzátor a interpret konfiguračního souboru
- stateless forwarding
- engine pro transformace a pseudo-proměnné
- RPC kontrolní API
- API časovače

Interní knihovny zahrnují:

- některé komponenty ze staršího jádra Kamailia
- vrstvy databázové abstrakce (DB API v1 a v2)
- management (MI) API
- engine pro statistiku

5.1.2 Moduly

Momentálně existuje přes 200 různých modulů, jejich načtením lze získat funkce jako:

- registrar a lokalizační management
- accounting, autorizace, autentizace
- operace pro text a regulární výrazy
- stateless odpovědi
- stateful zpracovávání - management SIP transakcí
- sledování SIP dialogů
- instant messaging a presence rozšíření
- podpora RADIUS a LDAP

- SQL a non-SQL databázové spoje
- MI a RPC transportní služby
- Enum, GeoIP API a CPL interpreter
- skrývání topologie a NAT přechody
- load-balancing a směrování na základě nejnižší ceny
- zpracovávání asynchronních SIP požadavků
- interaktivní debugger konfiguračního souboru
- Lua, Perl, Python a Java SIP Servlet nadstavby

[23]

5.2 Konfigurační soubor

Konfigurační soubor Kamailia využívá svůj vlastní skriptovací jazyk, jeho struktura je plně popsána v dokumentaci [24]. Zde budou popsány základní specifické prvky tohoto jazyka.

5.2.1 Struktura

Struktura konfiguračního souboru `kamailio.cfg` se dělí na 3 části:

- globální parametry (global parameters)
- nastavení modulů (modules settings)
- směrovací bloky (routing blocks)

Je doporučeno je udržet v tomto pořadí, ale není to ve všech případech nutné.

5.2.2 Generické prvky

Komentáře

```
#toto je nejcasteji pouzivany radkovy komentar
//taktez radkovy komentar
/*blokovy
komentar*/
```

Hodnoty - jsou 3 typy:

- integer - čísla reprezentována 32-bitovou hodnotou
- boolean - logická 1 (true, on, yes) nebo logická 0 (false, off, no)
- string - bloky textu ohraničené uvozovkami ("...", '...')

Identifikátory - nejsou ohraničeny uvozovkami a řídí se pravidly pro integer a boolean hodnoty. Jsou to například parametry a funkce jádra, funkce modulů, atd., například:

```
return
```

Proměnné - začínají znakem $, například:

```
$var(x) = $rU + "@" + $fd;
```

Akce - jsou to prvky užívané uvnitř směrovacích bloků zakončené znakem ";". Mohou vykonávat funkci z jádra nebo modulu, podmínku, smyčku, nebo přiřazovací výraz.

```
  sl_send_reply("404", "Not found");
  exit;
```

Výrazy - jsou to skupiny tvrzení, proměnných, funkcí, a operátorů

```
if(!t_relay())

if($var(x)>10)

"sip:" + $var(prefix) + $rU + "@" + $rd
```

5.2.3 Směrovací bloky

Směrovací bloky se narozdíl od ostatních částí konfiguračního souboru zpracovávají "za běhu". Při přirovnání k běžným programovacím jazykům je můžeme vnímat jako funkce. Směrovací blok je idendifikován specifickým označením následovaným jménem v hranatých závorkách. Následují akce ve složených závorkách.

```
route_block_id[NAME] {
  ACTIONS
}
```

Jméno může být tvořeno jakýmkoli alfanumerickým stringem, jehož text nemusí nutně ohraničen uvozovkami. Tyto bloky mohou být vykonávány na základě síťových událostí (např. přijetí SIP zprávy), časovačů, nebo událostí specifických pro moduly. Dále se mohou vyskytovat i tzv. subsměrovací bloky, vyvolané ze směrovacích bloků, podobně jako funkce. Struktura je zřejmá z následujícího příkladu.

```
request_route{
  ...
  route("test");
  ...
}

route["test"]{
  ...
}
```

request_route Tento směrovací blok je vykonáván pro každou SIP žádost – může být považován za ekvivalentní k funkci "main()"

route Takto se označují dříve zmíněné subsměrovací bloky. Mohou být "zavolány"z kteréhokoli bloku (dříve pouze z bloku "request_route"), ale je zásadní do nich vkládat akce validní pro kořenový blok. Jejich formát je totožný s běžnými směrovacími bloky s tím rozdílem, že vrací integer tomu bloku, který je zavolal (např. "return 0"). Tuto hodnotu lze uchovat přes `$rc` proměnné.

Vrácená hodnota subsměrovacího bloku je vyhodnocována následujícím způsobem:

- záporná hodnota vrací logickou 0 (false)
- 0 interpretuje `exit`
- kladná hodnota vrací logickou 1 (true)

reply_route Obsahuje akce, které se vykonají pro každou SIP odpověď, kterou Kamailio obdrží ze sítě. Dle RFC 3261 se v tomto bloku nesmí provádět žádné akce vykonávající směrování, ale může být použita akce `drop`. Tento blok je volitelný.

onreply_route Toto je směrovací blok vykonáván modulem texttttm. Obsahuje akce, které se vykonávají v kontextu aktivních transakcí.

onsend_route Je vykonáván pouze při předávání požadavků (forwarding). Mohou zde být kontrolovány parametry pro cílovou destinaci (IP adresy, porty, protokoly, apod.).

event_route Generický typ směrovacího bloku vykonávaný při specifických událostech. Jeho struktura je následující: `event_route[groupid:eventid]`

- groupid - mělo by být stejné jako jméno směrovacího bloku, ze kterého se vykoná
- eventid - nějaký smysluplný text popisující událost

[24]

5.3 Nástroje

5.3.1 kamctl

Tento nástroj je obsažen v základní instalaci. Jeho konfigurační soubor `kamctlrc` je umístěn ve stejné složce jako konfigurační soubor `kamailio.cfg`. Je potřeba jej upravit nastavením SIP domény a databázového enginu, ale většina ostatních parametrů má své deafultní hodnoty. Používá se k řízení kamailia z příkazového řádku a poskytuje širokou škálu možných operací a výpisů.

5.3.2 kamdbctl

Je obsažen v základní instalaci. Využívá stejnou konfiguraci jako `kamctl`. Může být využit k vytvoření a správě databázové struktury vyžadované Kamailiem. Toto by měla být jedna z prvních procedur po instalaci Kamailia. Nepoužívá se ovšem ke správě záznamů databázových tabulek, ale pouze ke správě databázové struktury a přístupu k ní.

5.3.3 kamcmd

Je obsažena v základní instalaci. `Kamdbctl` je aplikace, která je Kamailiu schopna posílat RPC příkazy z příkazového řádku. Vyžaduje, aby byl v Kamailiu načten `ctl` modul.

5.3.4 siremis

Webové rozhrání pro správu Kamailia psané v PHP. Dostupné na: `http://www.siremis.org`

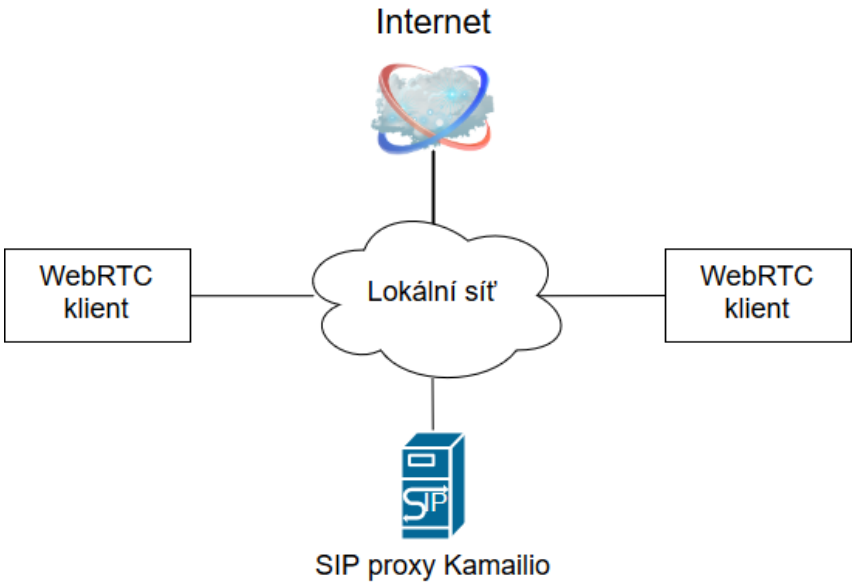
5.3.5 kamcli

Aplikace příkazového řádku psaná v pythonu. Je možnou alternativou k nástroji `kamctl` využívající modulární architekturu.
[25]

6 Nasazení SIP proxy Kamailia s WebRTC klienty

6.1 Pracovní topologie

Testovací topologie byla navrhnuta ze 3 entit, které jsou umístěny v jedné lokální síti a zároveň mají přístup k internetu. Přístup k internetu je nutný zaprvé ke stáhnutí WebRTC klientů a zadruhé kvůlu ICE protokolu pomocí kterého se klienti při zahajování hovoru budou snažit připojit ke vzdáleným serverům. Tyto 3 základní entity jsou logické povahy, tudíž mohou být umístěny na jediném zařízení, ale pro lepší rozlišení je lepší aby byla každá entita reprezentována samostatným zařízením.



Obrázek 10: Pracovní topologie

6.2 Instalace

Kamailio lze nainstalovat pouze na distribucích systému Linux. V této práci byla použita verze 5.1.2. Instalace obsahuje základní moduly a konfigurační soubory kamailio.cfg a kamctlrc umístěné ve složce `/etc/kamailio`. Moduly, které nejsou obsaženy v základní instalaci je třeba dodat podobným způsobem, a k nim knihovny, které mohou být případně vyžadovány. Instalace MySQL balíčků je nutná pro databázové funkce Kamailia. OpenSSL je vyžadováno TLS modulem.

```
# apt install kamailio kamailio-mysql-modules kamailio-tls-modules kamailio-
    websocket-modules
# apt install mysql-server mysql-client
# apt install libunistring-dev
# apt install openssl
```

6.3 Konfigurace Kamailia

V konfiguračním souboru `/etc/kamailio/kamctlrc` byly pouze odkomentovány následující řádky, a zvolena vlastní SIP doména (použita IP adresa). Řádek s parametrem `DBRWPW` byl odkomentován, aby se nemuselo zadávat heslo při zápisu a čtení z databáze (např. tvorba a výpis účtů).

```
SIP_DOMAIN = 192.168.2.224
DBENGINE = MYSQL
DBRWPW="kamailiorw"
```

[25]

Následuje konfigurace v souboru `/etc/kamailio/kamailio.cfg`. Úplně na začátek konfiguračního souboru byly vloženy definice pro adresy a porty, na kterých bude proxy poslouchat. Pro zabezpečené WebSocket spojení to je `192.168.2.224:4443`. Definice `WITH_MYSQL`, `WITH_AUTH`, a `WITH_TLS` aktivovaly kód již obsažený v dodané konfiguraci. Pro definici WITH_WEBSOCKETS byly přidány vlastní kusy kódu do příslušných konfiguračních bloků.

```
#!KAMAILIO
#!substdef "!MY_IP_ADDR!192.168.2.224!g"
#!substdef "!MY_DOMAIN!192.168.2.224!g"
#!substdef "!MY_WS_PORT!8080!g"
#!substdef "!MY_WSS_PORT!4443!g"
#!substdef "!MY_WS_ADDR!tcp:MY_IP_ADDR:MY_WS_PORT!g"
#!substdef "!MY_WSS_ADDR!tls:MY_IP_ADDR:MY_WSS_PORT!g"

#!define WITH_MYSQL
#!define WITH_AUTH
#!define WITH_WEBSOCKETS
#!define WITH_TLS
```

Tento kód byl vložen mezi globální parametry a udává, že proxy bude poslouchat na dříve definovaných adresách.

```
####### Global Parameters #########
listen = MY_IP_ADDR
#!ifdef WITH_WEBSOCKETS
   listen = MY_WS_ADDR
   #!ifdef WITH_TLS
      listen = MY_WSS_ADDR
   #!endif
#!endif
```

```
tcp_connection_lifetime = 3604
tcp_accept_no_cl = yes
tcp_rd_buf_size = 16384
```

Naspod sekce modulů pro byl vložen kód pro načtení modulů potřebných pro práci s websockety.

```
####### Modules Section ########
...
#!ifdef WITH_WEBSOCKETS
   loadmodule "xhttp.so"
   loadmodule "websocket.so"
   loadmodule "nathelper.so"
#!endif
```

[26]

Zde bylo potřeba upravit některé proměnné v daných modulech. Formát je následující: `modparam("název modulu", "název proměnné", "vkládaná informace")`. Do proměnné `config` modulu `tls` byla vložena cesta ke konfiguračnímu souboru pro TLS modul, který byl posléze vytvořen. [27]

```
# ----------------- setting module-specific parameters ---------------
...
#!ifdef WITH_TLS
   modparam("tls", "config", "/etc/kamailio/tls.cfg")
#!endif

#!ifdef WITH_WEBSOCKETS
   modparam ( "nathelper|registrar" , "received_avp" , "$avp(RECEIVED)" )
#!endif
```

Zde byl vložen kód specifický pro websockety na začátek směrovacího bloku `request_route`, tudíž bude uplatněn pro všechny příchozí SIP zprávy. Indikátor `nat_uac_test(64)` ověřuje, jestli je transportní protokol dané zprávy websocket. Následující část kódu uprauje údaje v lokalizační databázi tak, aby je bylo možno využít ke směrování. Je li metoda zprávy REGISTER, `fix_nated_register()` vytvoří URI skládající se z reálné IP adresy a portu odesílatele. Tuto URI uloží do proměnné `received` v lokalizační tabulce příslušného účtu a pošle v odpovědi 200 OK v parametru `received` hlavičky `Contact`. U jiných metod se použije `fix_nated_contact()`, který přepíše hlavičku `Contact` ve zprávě tak, aby obsahovala zdrojovou IP adresu a port. [28]

```
####### Routing Logic ########
request_route {
   route(REQINIT);
     #!ifdef WITH_WEBSOCKETS
         if ( nat_uac_test ( 64 ) ) {
         force_rport ( ) ;
            if ( is_method ( "REGISTER" ) ) {
             fix_nated_register ( ) ;
            } else {
               fix_nated_contact ( ) ;
               if ( ! add_contact_alias ( ) ) {
                  xlog ( "L_ERR" , "Error aliasing contact <$ct> \n " ) ;
                  sl_send_reply ( "400" , "Bad Request" ) ;
                  exit ;
               }
      }
   }
      if (!is_method("REGISTER"))
            t_on_reply("WS_REPLY");
      #!endif
   ...
```

Subsměrovací blok byl upraven pro chybový stav v rámci websocketů.

```
route[WITHINDLG] {
   if (!has_totag()) return;
      if (loose_route()) {
      #!ifdef WITH_WEBSOCKETS
            if ( $du == "" ) {
               if ( ! handle_ruri_alias ( ) ) {
                     xlog ( "L_ERR" , "Bad alias <$ru> \n " ) ;
                     sl_send_reply ( "400" , "Bad Request" ) ;
                     exit ;
               }
            }
      #!endif
      ...
```

Následující blok kódu byl vložen na konec souboru. Je důležitý pro WebSocketové SIP odpovědi, u kterých přidává parametr `alias`, který obsahuje skutečnou adresu klienta. Poslední

část slouží pro zpracovávání HTTP zpráv, což je u WebSocketů používáno pro úvodní HTTP handshake.

```
#!ifdef WITH_WEBSOCKETS
   onreply_route[WS_REPLY] {
      if (nat_uac_test(64)) {
      add_contact_alias();
      }
   }

   event_route[xhttp:request] {
      #moznost http odpovedi - vhodne napr. pro otestovani TLS spojeni
            #xhttp_reply("200", "OK", "text/html","<html><body>Received HTTP
                request to $hu from [$si:$sp] with protocol $proto</body></html>
                ");
         set_reply_close();
         set_reply_no_connect();

         if ($Rp != MY_WS_PORT
         #!ifdef WITH_TLS
               && $Rp != MY_WSS_PORT
         #!endif
         ) {
         xlog("L_WARN", "HTTP request received on $Rp\n");
         xhttp_reply("403", "Forbidden", "", "");
         exit;
         }

         xlog("L_DBG", "HTTP Request Received\n");

         if ($hdr(Upgrade)=~"websocket"
            && $hdr(Connection)=~"Upgrade"
            && $rm=~"GET") {
         if ($hdr(Host) == $null || !is_myself("sip:" + $hdr(Host))) {
               xlog("L_WARN", "Bad host $hdr(Host)\n");
               xhttp_reply("403", "Forbidden", "", "");
               exit;
         }
         }
         xhttp_reply("404", "Not found", "", "");
```

```
    }
#!endif
```

[26]

6.4 TLS

Aby se server mohl vůči klientovi ověřit, což je u WebRTC nezbytné, byl vytvořen vlastní certifikát a pár klíčů.

Nejdříve byla vytvořena hiearchie složek pro certifikační autoritu a v ní počítadlo (`index.txt`). Pomocí OpenSSL byl vytvořen certifikát certifikační autority platný 10 let. Při vytváření bylo nutné zadat vlastní heslo a některé údaje ohledně státu a organizace, email a název (common name). Tento název by měl být shodný s doménou serveru, popř. IP adresou.

```
/etc/kamailio# mkdir demoCA
/etc/kamailio# cd demoCA
/etc/kamailio/demoCA# mkdir newcerts
/etc/kamailio/demoCA# echo '01' > serial
/etc/kamailio/demoCA# touch index.txt
/etc/kamailio/demoCA# openssl req -new -x509 -extensions v3_ca -keyout key.pem
    -out cert.pem -days 3650
```

V druhé složce byl vytvořen pár soukromého a veřejného klíče a požadavek na podepsání certifikátu (CSR). Opět byly zadány popisné údaje, které musí být zadány stejně jako u příkazu výše.

```
/etc/kamailio/demoCA# cd ..
/etc/kamailio# mkdir mydomain
/etc/kamailio# cd mydomain/
/etc/kamailio/mydomain# openssl req -new -nodes -keyout key.pem -out req.pem
```

Posledním příkazem byl CSR podepsán certifikátem certifikační autority. Bylo zadáno heslo shodné s heslem zadaným při vytváření certifikátu certifikační autority.

```
/etc/kamailio/mydomain# cd ..
/etc/kamailio# openssl ca -days 730 -out mydomain/cert.pem -keyfile demoCA/key.
    pem -cert demoCA/cert.pem -infiles mydomain/req.pem
```

[29]

V konfiguračním souboru Kamailia byla nastavena cesta ke konfiguračnímu souboru modulu TLS (`tls.cfg`). Ten bylo potřeba vytvořit ve zvoleném adresáři a poté do něj vložit následující konfiguraci. Ta může být rozdělena do bloků, které jsou buď typu `server`, nebo typu `client`. Bloky typu `client` jsou platné pro příchozí spojení, bloky typu `server` pro odchozí. V této

práci je ovšem důležitá situace kdy bude Kamailio serverem, jelikož klient bude vždy zahajovat spojení zprávou typu REGISTER.

```
#### tls.cfg ####
[server:default] #1
method = TLSv1+ #2
verify_certificate = yes #3
require_certificate = no
private_key = /etc/kamailio/mydomain/key.pem #4
certificate = /etc/kamailio/mydomain/cert.pem
#ca_list = /etc/kamailio/demoCA/cert.pem

[client:default] #5
verify_certificate = yes
require_certificate = yes
```

1. Kromě toho, že jsou tyto bloky typu server nebo client, mohou mít za dvojtečkou vypsanou IP adresu a port. Pokud mají za dvojtečkou `default`, potom tento blok platí pro všechna spojení, která nesouhlasí s žádným jiným blokem.
2. Defaultní protokolová metoda je `TLSv1`. Není nutné tuto metodu měnit, ale univerzálnější je volba `TLSv1+`, která zároveň akceptuje i novější metody 1.1 a 1.2.
3. V tomto nastavení Kamailio nevyžaduje po klientech ověření certifikátem, ale pokud jej obdrží, tak jej i ověří.
4. Zde jsou zadané cesty k privátnímu klíči a certifikátu. V případě, že by bylo potřeba ověřovat klienty, nebo by Kamailio zastávalo roli klienta, nahrál by se certifikát certifikační autority přes parametr `ca_list`.
5. V případě, že by Kamailio zastávalo roli klienta, bude vyžadovat oveření certifikátem, a poté tento certifikát samo ověří.

[27]

6.5 Zprovoznění Kamailia

Před samotným spuštěním Kamailia musela být vytvořena databáze, kterou bude využívat.

```
# kamdbctl create
```

Následujícím příkazem se spouští Kamailio, popř. příkazy `stop` a `restart` slouží pro vypnutí a resetování. Při úspěšném spuštění následuje výpis přiřazeného PID (Proccess ID).

```
# kamctl start
```

Příkazem `kamctl add` se vytváří uživatelské účty. Není nutné, aby bylo spuštěné Kamailio.

```
# kamctl add 100 100
# kamctl add 200 200
```

Zobrazení informací o vytvořeném účtu lze provézt příkazem `kamctl show`, např.:

```
# kamctl show 100
```

Při neúspěšném spuštění lze zkontrolovat konfiguraci Kamailia příkazem:

```
# kamailio -c
```

Pokud problémy přetrvávají, přestože kontrola nevypsala žádnou chybu, lze využít pokročilejšího příkazu:

```
# kamailio -E -ddd
```

Někdy se při neúspěšném zpuštění nesprávně ukončí proces programu. V tom případě je vhodné najít proces kamailia například přes nástroj netstat a poté jej terminovat.

6.6 Konfigurace WebRTC klientů

V prvé řadě je nezbytné, aby daný prohlížeč podporoval technologii WebRTC. Mezi standardními prohlížeči WebRTC podporují současné verze prohlížečů Firefox, Chrome, Safari, Opera. U mobilních zařízení jsou to mobilní verze stejných prohlížečů, vyjma Opery Mini. Úplný seznam včetně porovnání jednotlivých verzí lze najít například na stránce `https://caniuse.com/#search=webrtc`. V základním nastavení prohlížeč neakceptuje dříve vytvořený TLS certifikát, jelikož není podepsaný veřejně známou certifikační autoritou, která by byla v databázi prohlížeče. V této situaci jsou dvě možností, jak postupovat:

1. Přidat IP adresu s portem serveru do seznamu vyjímek, čehož lze dosáhnout opět dvěmi způsoby:
   (a) Zadat do adresního pole IP adresu a port serveru pro protokol https: `https://192.168.2.224:4443`, a v rámci výstrahy udělit pro server vyjímku, typicky pomocí tlačítka na stránce.
   (b) Přidat server do seznamu vyjímek skrze nastavení prohlížeče, což může být specifičtější pro jednotlivé prohlížeče. Pro Mozillu Firefox je to "Options/Privacy & Security-/Security - Certificates - View Certificates/Servers - Add Exception". Zde se do textového pole zadá adresa serveru ve stejném formátu jako u předchozího způsobu.

2. Další sofistikovanější způsob je nahrát do prohlížeče certifikát certifikační autority, kterým byl podepsán certifikát, kterým se ověřuje server. Takový certifikát byl vytvořen při konfiguraci Kamailia a umístěn na `/etc/kamailio/demoCA/cert.pem`. Do prohlížeče je možné jej nahrát přes nastavení hned vedle udělování vyjímek – záložka "Authorities". Pro tuto metodu ovšem musela být zadána metadata při tvorbě certifikátů dle náležitostí.

Otestovány byly všechny z uvedených metod.

Pro jednoduchost byla zvolena dema dvou SIP WebRTC klientů:

- sipML5 od společnosti Doubango dostupné z `https://www.doubango.org/sipml5/call.htm`
- JsSIP dostupné z `https://tryit.jssip.net/`

Tito klienti jsou interoperabilní, tudíž byly v této práci otestováni oba v různých kombinacích. Jejich konfigurace je v zásadě podobná, ale názvy jednotlivých parametrů se liší.

sipML5

- Private Identity: 100
- Public Identity: sip:100@192.168.2.224
- Password: 100
- Realm: 192.168.2.224
- WebSocket Server URL: wss:192.168.2.224:4443

JsSIP

- SIP URI: sip:200@192.168.2.224
- SIP password: 200
- WebSocket URI: wss:192.168.2.224:4443

Po potvrzení konfigurace mezi klientem a serverem proběhl registrační dialog a mezi klienty bylo možné libovolně navázat hovor. Při registraci Kamailio vytvoří záznam v lokalizační databázi, tyto záznamy lze vypsat pomocí příkazu:

```
# kamctl ul show
```

7 Analýza provozu

WebRTC využívá pro signalizaci šifrované spojení, což prakticky znemožňuje přečíst obsah těchto zpráv prostým zachycením například Wiresharkem. Proto je jediným východiskem organizovat ukládání zpráv přímo skrze Kamailio nebo na straně klientů přes rozhraní prohlížeče ("pravé tračítko na stránce"/Inspect Element/Console). Nejjednodušším řešením by bylo je ukládat lokálně do záznamů, tzv. logů, a posléze je přečíst ze souboru. Mnohem flexibilnější a organizovanější řešení ovšem nabízí monitorovací server Homer.

7.1 Monitorovací server Homer

Homer je VoIP monitorovací server, který přijímá zejména SIP zprávy enkapsulované HEP protokolem, ukládá je do databáze a posléze k nim poskytuje přístup skrze webové rozhrání. Neumožňuje pouze přístup k datům, ale poskytuje i zpracování různých typů grafů a další analytické nástroje.

V tomto scénáři se vytváří další topologie zkládající se ze dvou entit:

- **Capture agent** - zachycuje a duplikuje SIP provoz, duplikáty enkapsuluje pomocí protokolu HEP, a posílá jej na capture server. Do HEP hlavičky ukládá dodatečná data o přenášené zprávě, která nemusí být jednoznačně zjistitelná z jejího obsahu, například adresy nebo časové známky. V tomto případě bude Capture agentem Kamailio, ale v jiných případech jím může být i Asterisk, FreeSwitch, CaptAgent apod.
- **Capture server** - přijímá a zpracovává enkapsulovaný SIP provoz, ukládá jej do databáze, pomocí webového rozhrání k němu poskytuje přístup atd.

Monitorovací server Homer je dostupný v několika verzích. Liší se zejména využitím různých technologií a uživatelským rozhraním. Základní z nich jsou:

- **Homer 3** - první již zastaralá verze
- **Homer 5** - současně nejpoužívanější, pro prozhrání využívá PHP
- **Homer 7** - nejnovější verze, která je stále ve vývoji; pro rozhrání využívá JavaScript

V topologii, která je výstupem této práce, byl nejdřív otestován Homer 5, ovšem při zachycování SIP zpráv metody INVITE a odpovědí 200 OK vyvstal problém, kdy Homer nebyl schopen tyto zprávy uložit do databáze. Tyto zprávy byly moc dlouhé (WebRTC v SDP zprávách posílá mnoho údajů) a nesplňovaly podmínku pro délku zprávy definovanou Homerem. Přes podniknuté zásahy do konfigurace a databáze se nepodařilo tento problém opravit, proto tato verze není do práce zahrnuta. Dále byl stejným způsobem otestován Homer 7, který byl funkční v požadovaných aspektech. Tato verze může být zkombinovaná s vícero technologiemi, takže byla v rámci této práce vybrána kombinace, která se zdála být nejtypičtější. Pro organizaci a běh Homeru je využit program Docker. Vybrané komponenty jsou označeny tučným písmem:

- API + UI
  - **NodeJS 8.x+**
- Databáze pro data (*vybrat jednu*)
  - **Postgres 9.x**
  - Elasticsearch
  - Loki
- Databáze časových řad (*vybrat jednu*)
  - **InfluxDB**
  - Prometheus
  - Elasticsearch
- HEP Capture Server (CS) - součástí vybrané instalace (*vybrat jeden*)
  - **HEPlify server** - vyvinutý v Go, ideální pro velké HEP zátěže
  - HEPop - vyvinutý v NodeJS, ideální pro RTC události a JSON streamy

[30]

Nejdříve byly nainstalovány balíčky dle vybrané konfigurace, Docker, a Git stahování ze serveru "github.com".

```
# apt install docker.io nodejs postgresql influxdb git
```

Ve libovolné složce stačí zadat příkaz pro stáhnutí instalačních souborů Homeru.

```
# git clone https://github.com/sipcapture/homer7-docker.git
```

Tyto soubory jsou organizované do "stromu"složek, ve kterém je cílová složka definována výběrem komponentů. V cílové složce je instalační soubor `docker-compose`. Kombinací s příkazem `up` se stáhnou všechny potřebné komponenty, a poté se sestaví výsledný monitorovací server. Příznak `-d` značí, že se terminál nezafixuje – bez tohoto příznaku se budou do terminálu kontinuálně vypisovat procesy Homeru. Stejná metoda slouží pro každé další spouštění.

```
# cd homer7-docker/heplify-server/hom7-hep-influx/
/homer7-docker/heplify-server/hom7-hep-influx# docker-compose up -d
```

Komponenty Homeru, které jsou spuštěny v kontejnerech Dockeru si lze vypsat příkazem:

```
# docker ps
```

Když je Homer server sestaven, dle výchozí konfigurace poslouchá na adrese `127.0.0.1` a portu `9060`. V případě Homeru 7 na tomto portu přijímá pouze zprávy enkapsulované protokolem HEP verze 3.

SIP proxy Kamailio, které je součástí této práce bude zastávat funkci capture agenta, neboli "Trace Node". V Kamailiu pro tuto funkci slouží modul SipTrace, proto byl potřeba opět upravit konfigurační soubor `kamailio.cfg`. V rámci náležitostí byla vytvořena definice `WITH_HOMER`

```
#!define WITH_HOMER
```

Na konec sekce modulů byl vložen příkaz pro načtení dříve zmíněného modulu.

```
####### Modules Section ########
...
#!ifdef WITH_HOMER
   loadmodule "siptrace.so"
#!endif
```

V části nastavení modulů je nejdůležitější položka `duplicate_uri`, kde byla v tomto případě vložena adresa a port, na které poslouchá monitorovací server Homer. V dané verzi Homeru musí být zapnutá HEP enkapsulace verze 3.

```
# ----------------- setting module-specific parameters ---------------
...
#!ifdef WITH_HOMER
   modparam("siptrace", "duplicate_uri", "sip:127.0.0.1:9060")
   modparam("siptrace", "hep_mode_on", 1)
   modparam("siptrace", "hep_version", 3)
   modparam("siptrace", "trace_to_database", 0)
   modparam("siptrace", "trace_flag", 22)
   modparam("siptrace", "trace_on", 1)
#!endif
```

Následujícím krokem je vložení příkazu `sip_trace();`, který dává povel k duplikaci zprávy v momentu, kdy Kamailio onen povel zpracuje, což závisí na umístění ve směrovacích blocích. Ideální v tomto případu je umístit jej na úplný začátek směrovacícho bloku `request_route`, takže budou duplikovány a vyslány všechny možné zprávy.

```
request_route {
   #!ifdef WITH_HOMER
           sip_trace();
        setflag(22);
   #!endif
   ...
```

Pro zachycení bezestavově směrovaných zpráv metody ACK byl přidán další směrovací blok (známá chyba tohoto modulu):

```
onsend_route {
    if (is_method("ACK")) {
        sip_trace();
    }
}
```

[31]

K webovému rozhraní Homeru se přistupuje pomocí webového prohlížeče dle výchozí konfigurace na adrese `127.0.0.1:9080` a přihlašovací údaje jsou *admin/sipcapture*. Základním parametrem, kterým se řídí většina funkcí je časové rozmezí. Po přihlášení je nastaven na "Today", takže pro hledání staršího záznamu je nutné jej změnit na požadovaný časový úsek, ovšem dle vlastního testování musí být v rámci jednoho dne. Rozhraní je plně nastavitelné, lze přidávat vlastní stránky, okna pro vyhledávání nebo grafy a upravovat jejich velikost a pozici. Nejdůležitější funkcí je ale samotné vyhledávání zpráv. Pro vyhledání požadovaných zpráv lze použít okno, které je součástí výchozího rozhraní, ale pro vyhledání např. registračních zpráv se musí změnit profil v nastavení okna – `call` pro signalizaci hovorů nebo `registration` pro registrace. Vyhledat lze buďto všechny zprávy v rámci daného profilu nebo například podle zdrojové/cílové IP adresy a portů. Po každé provedené změně nastavení je nutno uložit celé rozhraní příslušným tlačítkem v jeho pravém horním rohu. Vyhledané zprávy jsou zobrazeny v tabulce s parametry v jednotlivých sloupcích a rozlišeny barvami dle jednotlivých hovorů. Po vybrání hovoru jej lze zobrazit graficky a exportovat do formátu `.txt` nebo `.pcapng`.

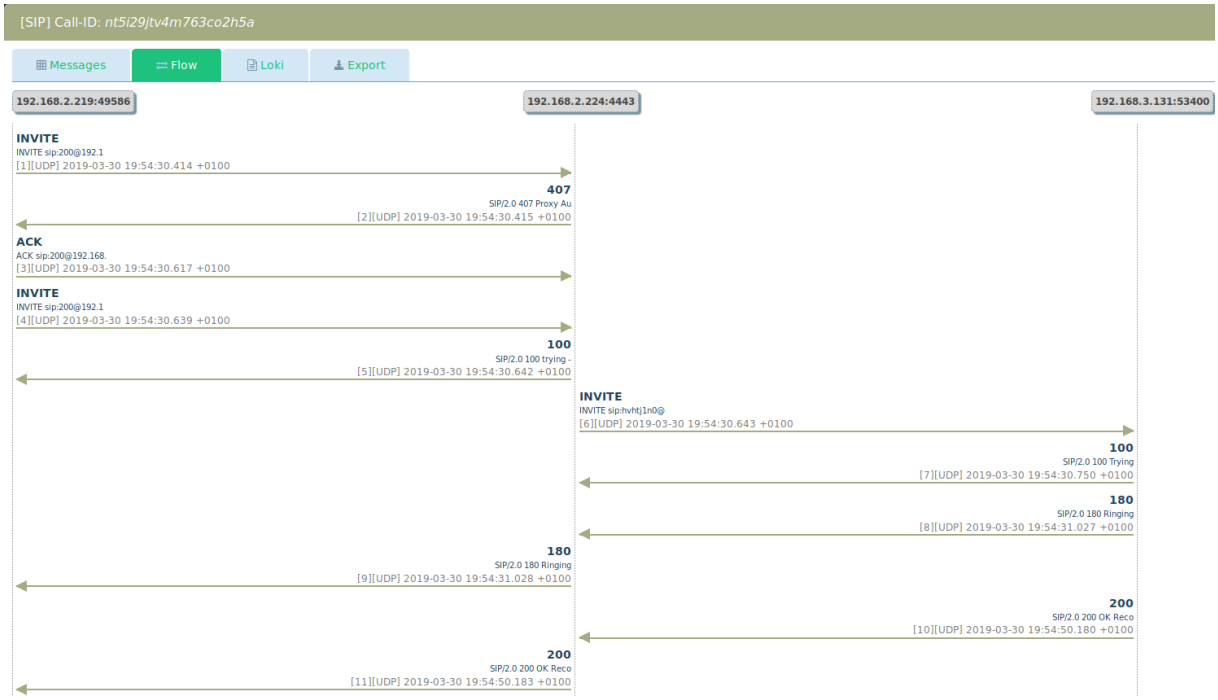
7.2 Zachycená SIP signalizace

SIP proxy Kamailio a 2 WebRTC klienti nakonfigurovaní dle předchozí kapitoly byli použiti pro vytvoření ukázky hovoru. Každá z těchto entit byla nasazena na různých zařízeních ve stejné síti, disponujících kamerou a mikrofonem. Kamailio, modifikované dle této kapitoly, posílalo všechny SIP zprávy na lokální monitorovací server Homer, na kterém byla signalizace hovoru posléze dohledána, graficky zobrazena (obr. 11) a exportována do formátu `.pcapng` (obr. 12). Pro ilustraci byla signalizace stejného hovoru zachycena v programu Wireshark přímo na síťovém rozhraní zařízení, na kterém bylo nasazeno Kamailio (obr. 13).

Před samotným hovorem klienti navázali zabezpečené WebSocket spojení se SIP proxy a registrovali se k dříve vytvořeným účtům. Porty si vybrali náhodně při zahájení spojení. Klient na adrese 192.168.2.219 se registroval k účtu 100, klient na adrese 192.168.3.131 k účtu 200. Na adrese 192.168.224 a portech 4443 a 5060 je SIP proxy Kamailio.

V grafickém zobrazení je vykreslena zdánlivě typická SIP komunikace s přidanou autentizací. Hovor zahajuje klient 100 zprávou metody INVITE, ta zahrnuje SDP tělo. Na tu přijde odpověď 180 Ringing značící vyzvánění a jakmile cílový účastník hovor přijme, vyšle volajícímu odpověď

200 OK obsahující SDP tělo. Tímto je zahájen hovor, a v jeho průběhu volaný posílá zprávy metody UPDATE až do momentu, kdy jeden z účastníků terminuje hovor a vyšle zprávu metody BYE. Zprávy metod ACK, UPDATE a BYE Kamailio sice přijímá na daném portu 4443, ale vyšle je z portu 5060, což sice není standardní chování, ale dle zachycených paketů je to záležitost pouze duplikovaného provozu - Kamailio pravěpodobně do HEP hlavičky ukládá nesprávný port.



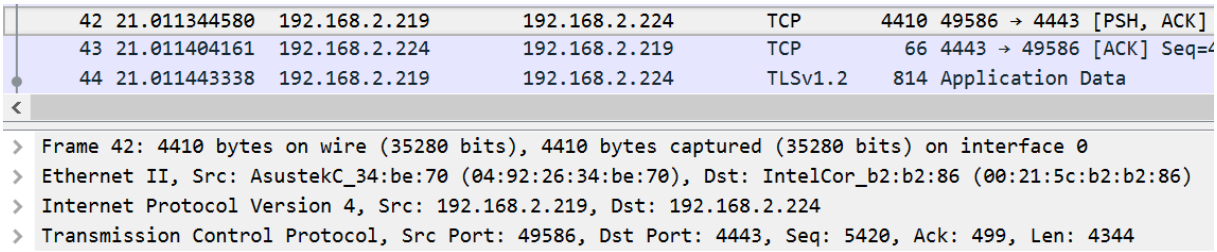
Obrázek 11: Ukázka grafického znázornění SIP komunikace v Homeru 7

```
Session Initiation Protocol (INVITE)
  Request-Line: INVITE sip:200@192.168.2.224 SIP/2.0
  Message Header
    Via: SIP/2.0/WSS 4t1q4p2m66rn.invalid;branch=z9hG4bK5330440
    Max-Forwards: 69
    To: <sip:200@192.168.2.224>
    From: "100" <sip:100@192.168.2.224>;tag=mjmd0v90ce
    Call-ID: nt5i29jtv4m763co2h5a
    CSeq: 5678 INVITE
    Proxy-Authorization: Digest algorithm=MD5, username="100", realm="192.168.2.224", nonce="XJ+8klyfu2bA1XWuKx5kT0m0RLKDoE1/"
    Contact: <sip:qggau7pf@4t1q4p2m66rn.invalid;transport=ws;ob>
    Content-Type: application/sdp
    Session-Expires: 90
    Allow: INVITE,ACK,CANCEL,BYE,UPDATE,MESSAGE,OPTIONS,REFER,INFO
    Supported: timer,ice,replaces,outbound
    User-Agent: JsSIP 3.2.15
    Content-Length: 4349
  Message Body
```

Obrázek 12: Ukázka SIP zprávy exportované z Homeru 7

Když je signalizace zachycena v průběhu hovoru Wiresharkem, nelze z obsahu těchto zpráv vyčíst prakticky nic, kromě celkové délky, IP adres, portů, atd. Stojí za zmínku, že verze proto-

kolu TLS je 1.2.



Obrázek 13: Ukázka reálné komunikace ve Wiresharku

Jakmile bylo v pracovní síti nasazeno Kamailio s uvedenou konfigurací, bylo možné v dané síti libovolně realizovat testovací hovory mezi libovolnými zařízeními bez dodatečné instalace a s minimální konfigurací. V těchto testovacích hovorech se jako problémový mechanismus projevil ICE protokol, který se snažil o spojení s internetovými servery a v některých případech způsoboval zpoždění při navazování hovoru nebo úplnou neschopnost hovor navázat. Toto by pravděpodobně mohlo být vyřešeno při nasazení vlastní WebRTC SIP aplikace.

Při zachytávání WebRTC SIP provozu byla u zavedeného monitorovacího serveru Homer verze 5 znát neuzpůsobenost pro tento typ komunikace a prokázal se jako problémový. Na ony nedostatky je u vývoje nové verze 7 brán ohled a nabízí řadu dalších technologií uzpůsobených real-time komunikaci. Proto se zdá být v tomto případě verze 7 vhodnější, ačkoli stále vykazuje některé znaky probíhajícího vývoje.

8 Závěr

V této práci byla popsána později prezentovaná komunikace z různých pohledů na aplikační i signalizační procesy a funkce. Byl kladen veliký důraz zejména na popis signalizačního protokolu SIP, který byl posléze stěžejním prvkem praktické části. Naproti tomu WebRTC aplikace byla již předem definovaná použitými demy, tudíž v byla praktické části "schovaná"v prohlížeči a předmětné bylo jen uživatelské nastavení. Přesto šlo do jisté míry pozorovat to co bylo předem popsáno.

Navrhnuta byla univerzální topologie situovaná v lokální síti s pevně daným SIP proxy serverem Kamailio a alespoň dvěmi WebRTC klienty s cílem realizovat testovací WebRTC SIP hovor. Pro tuto topologii byl nakonfigurován SIP proxy server Kamailio dle požadavků kladených WebRTC modelem, to jest zabezpečení pomocí protokolu TLS a zpracovávání WebSocket spojení a WebSocket SIP subprotokolu. Na základě těchto konfigurací společně s konfigurací klientů byla daná topologie zprovozněna a bylo možné detekovat šifrovanou komunikaci mezi klienty a serverem. Aby bylo možné analyzovat tuto komunikaci, uznalo se za vhodné přidat do této topologie monitorovací server Homer, který poskytuje přehledné rozhrání pro zobrazení SIP komunikace a jejích zpráv. Po testování různých verzí Homeru se přes drobné nedostatky osvědčila jeho nejnovější verze 7, která je stále ve vývoji. Nezbytně se dodala další konfigurace Kamailia a nakonec bylo možné názorně zobrazit zachycenou SIP komunikaci v rozhrání Homeru a přečíst její jednotlivé zprávy.

Pro zapracování do výuky IP telefonie byla vytvořena příloha, ve které je zaznamenána konfigurace obsažena v této práci, společně s instrukcemi k jejímu zprovoznění.

Literatura

[1] HART Justin. *WebRTC For Dummies*. New York: John Wiley & Sons, 2015. ISBN 978-1-119-09878-2.

[2] BARZ B. Hans, BASSET A. Gregory. *Multimedia Networks*. UK: John Wiley & Sons Ltd, 2016, 1. edice. ISBN: 9781119090137

[3] VALIN JM., BRAN C.. *WebRTC Audio Codec and Processing Requirements* [online]. Request for Comments 7874, Květen 2016. Dostupné z IETF: https://tools.ietf.org/html/rfc7874

[4] ROACH A. B. *WebRTC Video Processing and Codec Requirements* [online]. Request for Comments 7742, Březen 2016. Dostupné z IETF: https://tools.ietf.org/html/rfc7742

[5] BERGKVIST Adam, Burnett C. Daniel, JENNINGS Cullen, et al. *WebRTC 1.0: Real-time Communication Between Browsers* [online]. W3C Candidate Recommendation 7742, March 2016. Dostupné z W3C: https://www.w3.org/TR/2018/CR-webrtc-20180927/

[6] BAUGHER M., MCGREW D., NASLUND M., et al. *The Secure Real-time Transport Protocol (SRTP)* [online]. Request for Comments 3711, March 2004. Dostupné z IETF: https://tools.ietf.org/html/rfc3711

[7] RESCORLA E., MODADUGU N. *Datagram Transport Layer Security Version 1.2* [online]. Request for Comments 6347, January 2012. Dostupné z IETF: https://tools.ietf.org/html/rfc6347

[8] SCHULZRINNE H., CASNER S., FREDERICK R., JACOBSON V. *RTP: A Transport Protocol for Real-Time Applications* [online]. Request for Comments 7742, July 2003. Dostupné z IETF: https://tools.ietf.org/html/rfc3550

[9] LENNOX J., WESTERLUND M., WU Q. PERKINS C.. *Sending Multiple RTP Streams in a Single RTP Session* [online]. Request for Comments 8108, Březen 2017. Dostupné z IETF: https://tools.ietf.org/html/rfc8108

[10] KERANEN A., HOMBERG C., ROSENBERG J. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal* [online]. Request for Comments 5398, Červenec 2018. Dostupné z IETF: https://tools.ietf.org/html/rfc8445

[11] MAHY R., MATTHEWS P., ROSENBERG J. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)* [online]. Request for Comments 5366, Duben 2010. Dostupné z IETF: https://tools.ietf.org/html/rfc5766

[12] ROSENBERG J., MAHY R., MATTHEWS P., WING D. *Session Traversal Utilities for NAT (STUN)* [online]. Request for Comments 5398, Listopad 2008. Dostupné z IETF: https://tools.ietf.org/html/rfc5389

[13] LORETO Salvatore, ROMANO Pietro Salvatore. *Real-Time Communication with WebRTC*. Sebastopol: O'Reilly Media Ltd, 2014. ISBN: 9781449371876

[14] ALVESTRAND H. *Transports for WebRTC* [online]. Draft, October 2016. Dostupné z IETF: https://tools.ietf.org/html/draft-ietf-rtcweb-transports-17

[15] ALVESTRAND H. *Overview: Real Time Protocols for Browser-based Applications* [online]. Draft, Prosinec 2011. Dostupné z IETF: https://tools.ietf.org/html/draft-ietf-rtcweb-overview-19

[16] ROSENBERG J., SCHULZRINNE H., CAMARILLO G., et al. *SIP: Session initiation protocol* [online]. Request for Comments 3261, Červen 2002. Dostupné z IETF: https://tools.ietf.org/html/rfc3261

[17] JANAK Jan. *SIP Proxy Server Effectiveness*. Praha: Česká Technická Univerzita - Fakulta elektrotechniky - Oddělení počítačových věd, 2003.

[18] VOZNAK Miroslav. *Technologie a protokoly multimediálních komunikací pro integrovanou výuku VUT a VŠB-TUO*[CD-ROM]. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2014. ISBN 978-80-248-3326-2.

[19] HICKSON Ian. *The WebSocket API* [online]. W3C Candidate Recommendation 20, September 2012. Dostupné z W3C: https://www.w3.org/TR/websockets/

[20] GRIGORIK Ilya. *High Performance Browser Networking*. Sebastopol: O'Reilly Media Ltd, 2013. ISBN: 9781449344764

[21] FETTE I., MELNIKOV A. *The Websocket protokol* [online]. Request for Comments 6455, Prosinec 2011. Dostupné z IETF: https://tools.ietf.org/html/rfc6455

[22] CASTILLO Baz I., VILLEGAS Millan J., PASCUAL V. *The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)* [online]. Request for Comments 7118, January 2014. Dostupné z IETF: https://tools.ietf.org/html/rfc7118

[23] MIERLA Daniel-Constantin, MODROIU Elena-Ramona. *Kamailio SIP Server v3.2.0 Development Guide* [online]. ASIPTO, 2011. Dostupné z Asipto: http://www.asipto.com/pub/kamailio-devel-guide/

[24] Kamailio. *Kamailio SIP Server v5.3.x (devel): Core Cookbook* [online]. Kamailio, 2019. Dostupné z Kamailio SIP Server Wiki: https://www.kamailio.org/wiki/cookbooks/devel/core

[25] Kamailio. *Kamailio - Getting Started Guide* [online]. Kamailio, 2015. Dostupné z Kamailio SIP Server Wiki: https://www.kamailio.org/wiki/tutorials/getting-started/main

[26] DUNKLEY Peter. *WebSocket Module* [online]. Kamailio. Dostupné z Kamailio SIP Server Dokumentace: https://www.kamailio.org/docs/modules/stable/modules/websocket.html

[27] PELINESCU-ONCIUL Andrei, BOCK Carsten, JOHANSSON E. Olle. *TLS Module* [online]. Kamailio. Dostupné z Kamailio SIP Server Dokumentace: https://kamailio.org/docs/modules/devel/modules/tls.html

[28] SOBOLEV Maxim, HEINANEN Juha. *Nathelper Module* [online]. Kamailio, 5.x. Dostupné z Kamailio SIP Server Dokumentace: https://www.kamailio.org/docs/modules/5.0.x/modules/nathelper.html

[29] Kamailio. *Creating Certificates with OpenSSL* [online]. Kamailio, 2010. Dostupné z Kamailio SIP Server Wiki: https://www.kamailio.org/dokuwiki/doku.php/tls:create-certificates

[30] Sipcapture. *HOMER Seven* [online]. Sipcapture, 2018. Dostupné z Gitu: https://github.com/sipcapture/homer/tree/homer7

[31] MIERLA Daniel-Constantin. *SipTrace Module* [online]. Kamailio. Dostupné z Kamailio SIP Server Dokumentace: https://www.kamailio.org/docs/modules/devel/modules/siptrace.html

A Zaznamenaná konfigurace s instrukcemi ke zprovoznění

A.1 SIP proxy Kamailio

A.1.1 Instalace

Nejprve je potřeba nainstalovat Kamailio s požadovanými moduly, MySQL a OpenSSL. Libunistring je balíček vyžadovaný modulem.

```
# apt install kamailio kamailio-mysql-modules kamailio-tls-modules kamailio-
    websocket-modules
# apt install mysql-server mysql-client
# apt install libunistring-dev
# apt install openssl
```

A.1.2 Konfigurace

Konfigurační soubory Kamailia jsou umístěny v adresáři `/etc/kamailio/`.

V souboru `kamctlrc` se odkomentují následující řádky a určí doména.

```
SIP_DOMAIN = 192.168.2.224
DBENGINE = MYSQL
DBRWPW="kamailiorw"
```

Následující konfigurace se vkládá do souboru `kamailio.cfg`. Celý konfigurační soubor je rozdělen do sekcí, toto rozdělení je doporučeno dodržovat. Na začátku se nastavuje IP adresa vlastního zařízení a porty, na kterých bude Kamailio poslouchat.

```
#!KAMAILIO
#!substdef "!MY_IP_ADDR!192.168.2.224!g"
#!substdef "!MY_DOMAIN!192.168.2.224!g"
#!substdef "!MY_WS_PORT!8080!g"
#!substdef "!MY_WSS_PORT!4443!g"
#!substdef "!MY_WS_ADDR!tcp:MY_IP_ADDR:MY_WS_PORT!g"
#!substdef "!MY_WSS_ADDR!tls:MY_IP_ADDR:MY_WSS_PORT!g"

#!define WITH_MYSQL
#!define WITH_AUTH
#!define WITH_WEBSOCKETS
#!define WITH_TLS
```

Zde se aktivují dříve definované adresy.

```
####### Global Parameters #########
listen = MY_IP_ADDR
#!ifdef WITH_WEBSOCKETS
   listen = MY_WS_ADDR
   #!ifdef WITH_TLS
      listen = MY_WSS_ADDR
   #!endif
#!endif

tcp_connection_lifetime = 3604
tcp_accept_no_cl = yes
tcp_rd_buf_size = 16384
```

Na konec sekce modulů se vloží kód pro nahrání požadovaných modulů.

```
####### Modules Section ########
...
#!ifdef WITH_WEBSOCKETS
   loadmodule "xhttp.so"
   loadmodule "websocket.so"
   loadmodule "nathelper.so"
#!endif
```

U parametrů modulů se zadá zejména cesta ke konfiguračnímu souboru modulu TLS, který bude vytvořen později.

```
# ----------------- setting module-specific parameters ---------------
...
#!ifdef WITH_TLS
   modparam("tls", "config", "/etc/kamailio/tls.cfg")
#!endif

#!ifdef WITH_WEBSOCKETS
   modparam ( "nathelper|registrar" , "received_avp" , "$avp(RECEIVED)" )
#!endif
```

Zde stačí upravit začátek hlavního směrovacího bloku `request_route`.

```
####### Routing Logic ########
request_route {
   route(REQINIT);
     #!ifdef WITH_WEBSOCKETS
```

```
        if ( nat_uac_test ( 64 ) ) {
        force_rport ( ) ;
           if ( is_method ( "REGISTER" ) ) {
            fix_nated_register ( ) ;
           } else {
              fix_nated_contact ( ) ;
              if ( ! add_contact_alias ( ) ) {
                 xlog ( "L_ERR" , "Error aliasing contact <$ct> \n " ) ;
                 sl_send_reply ( "400" , "Bad Request" ) ;
                 exit ;
              }
     }
  }
     if (!is_method("REGISTER"))
          t_on_reply("WS_REPLY");
     #!endif
  ...
```

Opět se jedná o úpravu začátku směrovacího bloku.

```
route[WITHINDLG] {
  if (!has_totag()) return;
     if (loose_route()) {
     #!ifdef WITH_WEBSOCKETS
          if ( $du == "" ) {
             if ( ! handle_ruri_alias ( ) ) {
                  xlog ( "L_ERR" , "Bad alias <$ru> \n " ) ;
                  sl_send_reply ( "400" , "Bad Request" ) ;
                  exit ;
             }
          }
     #!endif
     ...
```

Tento blok kódu se jednoduše vloží na konec souboru.

```
#!ifdef WITH_WEBSOCKETS
  onreply_route[WS_REPLY] {
     if (nat_uac_test(64)) {
     add_contact_alias();
     }
```

```
    }

    event_route[xhttp:request] {
        #moznost http odpovedi - vhodne napr. pro otestovani TLS spojeni
            #xhttp_reply("200", "OK", "text/html","<html><body>Received HTTP
                request to $hu from [$si:$sp] with protocol $proto</body></html>
                ");
            set_reply_close();
            set_reply_no_connect();

            if ($Rp != MY_WS_PORT
            #!ifdef WITH_TLS
                  && $Rp != MY_WSS_PORT
            #!endif
            ) {
            xlog("L_WARN", "HTTP request received on $Rp\n");
            xhttp_reply("403", "Forbidden", "", "");
            exit;
            }

            xlog("L_DBG", "HTTP Request Received\n");

            if ($hdr(Upgrade)=~"websocket"
               && $hdr(Connection)=~"Upgrade"
               && $rm=~"GET") {
            if ($hdr(Host) == $null || !is_myself("sip:" + $hdr(Host))) {
                  xlog("L_WARN", "Bad host $hdr(Host)\n");
                  xhttp_reply("403", "Forbidden", "", "");
                  exit;
            }
            }
            xhttp_reply("404", "Not found", "", "");
    }
#!endif
```

A.1.3 Konfigurace TLS modulu

Nejdříve je potřeba vytvořit hiearchii složek pro certifikační autoritu a v ní počítadlo (`index.txt`). Pomocí openssl je vytvořen certifikát certifikační autority platný 10 let. Při vytváření je nutné

zadat vlastní heslo, a některé údaje ohledně státu a organizace, email a název (common name). Tento název by měl být shodný s IP adresou.

```
/etc/kamailio# mkdir demoCA
/etc/kamailio# cd demoCA
/etc/kamailio/demoCA# mkdir newcerts
/etc/kamailio/demoCA# echo '01' > serial
/etc/kamailio/demoCA# touch index.txt
/etc/kamailio/demoCA# openssl req -new -x509 -extensions v3_ca -keyout key.pem
    -out cert.pem -days 3650
```

V druhé složce se vytvoří pár soukromého a veřejného klíče a požadavek na podepsání certifikátu (CSR). Opět se musí zadat popisné údaje, které musí být stejné, jako u příkazu výše.

```
/etc/kamailio/demoCA# cd ..
/etc/kamailio# mkdir mydomain
/etc/kamailio# cd mydomain/
/etc/kamailio/mydomain# openssl req -new -nodes -keyout key.pem -out req.pem
```

Posledním příkazem se CSR podepisuje certifikátem certifikační autority. Zadané heslo musí být shodné s heslem zadaným při vytváření certifikátu certifikační autority.

```
/etc/kamailio/mydomain# cd ..
/etc/kamailio# openssl ca -days 730 -out mydomain/cert.pem -keyfile demoCA/key.
    pem -cert demoCA/cert.pem -infiles mydomain/req.pem
```

V konfiguračním souboru Kamailia byla nastavena cesta ke konfiguračnímu souboru modulu TLS (`tls.cfg`). Ten je třeba vytvořit ve zvoleném adresáři a poté do něj vložit následující konfiguraci.

```
#### tls.cfg ####
[server:default]
method = TLSv1+
verify_certificate = yes
require_certificate = no
private_key = /etc/kamailio/mydomain/key.pem
certificate = /etc/kamailio/mydomain/cert.pem
#ca_list = /etc/kamailio/demoCA/cert.pem

[client:default]
verify_certificate = yes
require_certificate = yes
```

A.1.4 Zprovoznění Kamailia

Před samotným spuštěním Kamailia musí vytvořit databáze, kterou bude Kamailio využívat.

```
# kamdbctl create
```

Následujícím příkazem se spouští Kamailio. Při úspěšném spuštění následuje výpis přiřazeného PID (Proccess ID).

```
# kamctl start
```

Příkazem kamctl add se vytváří uživatelské účty. Není nutné, aby bylo spuštěné Kamailio.

```
# kamctl add 100 100
# kamctl add 200 200
```

Zobrazení informací o vytvořeném účtu lze provézt příkazem kamctl show, např.:

```
# kamctl show 100
```

Při neúspěšném spuštění lze zkontrolovat konfiguraci Kamailia příkazem:

```
# kamailio -c
```

Pokud problémy přetrvávají, přestože kontrola nevypsala žádnou chybu, lze využít pokročilejšího příkazu:

```
# kamailio -E -ddd
```

Někdy se při neúspěšném zpuštění nesprávně ukončí proces programu. V tom případě je vhodné najít proces kamailia například přes nástroj netstat a poté jej terminovat.

A.2 Konfigurace WebRTC klientů

Vhodnými prohlížeči jsou např. Firefox nebo Chrome, na mobilních zařízení s operačním systémem Android je to pouze Firefox. V základním nastavení prohlížeč neakceptuje dříve vytvořený TLS certifikát, jelikož není podepsaný veřejně známou certifikační autoritou, která by byla v databázi prohlížeče. V této situaci je několik možností, jak postupovat:

1. Přidat IP adresu s portem serveru do seznamu vyjímek, čehož lze dosáhnout opět 2 způsoby:
   (a) Zadat do adresního pole IP adresu a port serveru pro protokol https, např. `https://192.168.2.224:4443`, a v rámci výstrahy udělit pro server vyjímku, typicky pomocí tlačítka na stránce.

(b) Přidat server do seznamu vyjímek skrze nastavení prohlížeče, což může být specifičtější pro jednotlivé prohlížeče. Pro Mozillu Firefox je to: "Options/Privacy & Security/Security - Certificates - View Certificates/Servers - Add Exception". Zde se do textového pole zadá adresa serveru ve stejném formátu jako u předchozího způsobu.

2. Další, sofistikovanější způsob je nahrát do prohlížeče certifikát certifikační autority, kterým byl podepsán certifikát, kterým se ověřuje server. Takový certifikát byl vytvořen při konfiguraci Kamailia a umístěn na: "/etc/kamailio/demoCA/cert.pem". Do prohlížeče je jej možno nahrát přes nastavení hned vedle udělování vyjímek – záložka "Authorities".

Jsou dostupné 2 dema klientů, která lze kombinovat.

- sipML5 od společnosti Doubango dostupné z `https://www.doubango.org/sipml5/call.htm`.
- JsSIP dostupné z `https://tryit.jssip.net/`.

Dále jsou uvedeny příklady konfigurace.

sipML5

- Private Identity: 100
- Public Identity: sip:100@192.168.2.224
- Password: 100
- Realm: 192.168.2.224
- WebSocket Server URL: wss:192.168.2.224:4443

JsSIP

- SIP URI: sip:200@192.168.2.224
- SIP password: 200
- WebSocket URI: wss:192.168.2.224:4443

A.3 Zachycení komunikace pomocí monitorovacího serveru Homer 7

A.3.1 Instalace

Mimo Homeru je potřeba nainstalovat Docker a několik dalších aplikací.

```
# apt install docker.io nodejs postgresql influxdb git
```

Do libovolné složky stačí zadat nástledující příkaz pro stažení instalačních souborů Homeru z Gitu.

```
# git clone https://github.com/sipcapture/homer7-docker.git
```

Instalace a spouštění se provádí jednotným způsobem v požadovaném adresáři.

```
# cd homer7-docker/heplify-server/hom7-hep-influx/
homer7-docker/heplify-server/hom7-hep-influx# docker-compose up -d
```

Probíhající procesy uvnitř kontejnerů Dockeru lze zobrazit příkazem:

```
# docker ps
```

A.3.2 Konfigurace Kamailia

Pro duplikování a posílání zpráv je potřeba dodatečná konfigurace Kamailia, která probíhá obdobně, jako v předchozím případě.

```
#!define WITH_HOMER
```

Pro duplikaci a enkapsulaci zpráv v Kamailiu slouží modul `siptrace`.

```
####### Modules Section ########
...
#!ifdef WITH_HOMER
loadmodule "siptrace.so"
#!endif
```

V nastavení modulů se zadá zejména adresa, na kterou se budou duplikované zprávy posílat, povolení protokolu HEP a jeho verze.

```
# ------- setting module-specific parameters -------

#!ifdef WITH_HOMER
   modparam("siptrace", "duplicate_uri", "sip:127.0.0.1:9060")
   modparam("siptrace", "hep_mode_on", 1)
   modparam("siptrace", "hep_version", 3)
   modparam("siptrace", "trace_to_database", 0)
   modparam("siptrace", "trace_flag", 22)
   modparam("siptrace", "trace_on", 1)
#!endif
```

Opět se upraví začátek hlavního směrovacího bloku:

```
request_route {
   #!ifdef WITH_HOMER
            sip_trace();
         setflag(22);
   #!endif
   ...
```

Pro duplikaci bezestavově směrovaných zpráv metody ACK je potřeba vytvořit nový směrovací blok.

```
onsend_route {
    if (is_method("ACK")) {
        sip_trace();
    }
}
```

K rozhrání Homeru se přistupuje pomocí webového prohlížeče dle výchozí konfigurace na adrese `127.0.0.1:9080` a přihlašovací údaje jsou *admin/sipcapture*.

B Přiložené CD

CD obsahuje:

- konfigurační soubory Kamailia
- zachycenou komunikaci
- dokument této práce